OK DTIC

ISG Report 104

IMAGE UNDERSTANDING RESEARCH

Final Technical Report

Covering Research Activity During the Period
October 1, 1982 through September 30, 1983

R. Nevatia
Editor
(213) 743-5516

Intelligent Systems Group
Departments of Electrical Engineering
and Computer Science
University of Southern California
Los Angeles, California 90089-0272

October 19, 1983

This research was supported by the Defense Advanced Research Projects Agency and was monitored by the Air Force Wright Aeronautical Laboratories under Contract F-33615-82-K-1786, DARPA order No. 3119.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISG Report 104	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) IMAGE UNDERSTANDING RESEARCH		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 10/01/82 - 09/30/83 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Principal Investigator Ramakant Nevatia		8. CONTRACT OR GRANT NUMBER(s) F33615-82-K-1786
9. PERFORMING ORGANIZATION NAME AND ADDRESS Departments of Electrical Engineering and Computer Science University of Southern California, L.A., CA 90089		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DARPA Order No. 3119
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE October 19, 1983 13. NUMBER OF PAGES 176
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Wright Aeronautical Laboratories Wright-Patterson Air Force Base Dayton, OH 45433		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Public

Approved for release: distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image Understanding, scene analysis, image segmentation, image matching, texture analysis, shadow analysis, VLSI

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This technical report summarizes the image understanding activities performed by USC during the period October 1, 1982 through September 30, 1983 under contract number F33615-82-K-1786 with the Defense Advanced Research Projects Agency, Information Processing Techniques Office. This contract is monitored by the Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Dayton, OH.

OVER

(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The purpose of this research program is to develop techniques and systems for understanding images, particularly for mapping applications. This report describes techniques for: visual inspection using linear features, relaxation matching for map-matching, techniques for contour matching, a segment-based stereo matching method, use of shadows in the interpretation of aerial images, use of image shading to infer 3-D depth, use of texture for image segmentation, VLSI implementation of graph isomorphism algorithms and a study of suitable architectures for implementing image understanding algorithms.

TABLE OF CONTENTS

SECTION 1

OVERVIEW AND SUMMARY

Ramakant Nevatia

1.1 INTRODUCTION

This report describes in detail our research under contract F-33615-82-K-1786 during the period of October 1 1982 to September 30, 1983 and constitutes the final report along with a previously published semi-annual progress report [1]. During this period we have concentrated on the following tasks: symbolic matching for a variety of applications, 3-D surface inference from 2-D images by using stereo, shadows and shading, texture based segmentation and parallel, VLSI implementations of image understanding algorithms. This section summarizes our results in these areas.

1.2 MATCHING

We have been developing a number of symbolic description matching algorithms that have been described previously in [1]; we have applied them to a variety of aerial image tasks previously. In section 2 of this report, Kaufmann, Medioni and Nevatia describe the application of our line matching algorithm to the task of inspecting a printed circuit board assembly for defects such as a missing or misplaced component. The generality of out technique is demonstrated by showing applicability to domains as diverse as aerial images and industrial inspection.

In section 3, Price gives a comparison of various relaxation matching techniques that use different criteria for updating matching confidences at each iteration. The comparison is in terms of efficiency and accuracy. In section 4, Price gives an algorithm for contour matching that is, in some sense, a combination of line matching and relaxation matching, and has been successfully applied to scenes of simple objects such as hand tools. In this case, the objects are on a bright light-table and thus a perfect segmentation is obtained.

1.3 STEREO

In section 5, Medioni and Nevatia describe a new stereo matching algorithm using line segments, rather than edges or area correlation, as has been common in previous work. Line segments are more global and also, in principle, lead to more efficient matching. We have developed a new matching criterion called "minimal differential disparity" that seems to lead to good global matches. We have successfully tested our algorithm on a number of difficult scenes. However, this work is still to be regarded only as a promising first step.

1.4 SHADOWS AND SHADING

Shadow and smoothly varying intensity, known as shading, are two important cues for inferring 3-D information from monocular images. The difficult task in shadow analysis is the correspondence of object boundaries with shadow boundaries. Previously, we developed a method that used a priori knowledge of specific shapes expected to be seen, e.g. rectangular buildings [2]. In section 6, Huertas presents a more general scheme relying on labeling of edges in images and good results are obtained for estimating heights of objects such as oil tanks. Our analysis indicates that shadow correspondence problem could be extremely hard in a general case, but that useful information can be obtained for the case of aerial images where shadows are often cast on relatively flat ground. In aerial images, sometimes there is no cue other than shadows that is applicable.

Smooth variations in shading are postulated as another cue for estimating 3-D surfaces from 2-D images. There has been much interest in this area in the last few years, but the application has been to extremely simple and highly controlled environments. We have empirically tested two of the leading methods with results given in section 7. Our results indicate that the techniques work very well for synthetic images but give disappointing results for natural images. We believe that these methods need further investigation due to the important of this cue.

1.5 SEGMENTATION

Segmentation in presence of texture has been a long standing problem in image understanding. We have, under previous contracts, developed some texture analysis methods that are new being applied to the segmentation problem. In section 8, Lee describes a method using "pyramids", with the expectation that at some level of pyramid, a textured region appears uniform and can be extracted easily. Another concern is with refining the boundary location. This method is still being fully developed and will be described in a forthcoming Ph.D. thesis.

In section 9, Weber and Sawchuk take a slightly different approach, using texture classification for segmentation. They also use windows of multiple sizes and choose the size that seems to give the highest confidence classification. This method also needs further development.

1.6 PARALLEL AND VLSI IMPLEMENTATIONS

It is becoming increasingly clear that complex IU algorithms can not be run in real-time on even very fast, conventional, simple-processor, general purpose machines. In fact, the limitations of computation power inhibit research into more powerful algorithms that are essential for improved performance. A part of our effort has been devoted to the study of parallel architectures suited for IU and construction of special VLSI devices for specific algorithms.

In section 10, Moldovan describes a parallel architecture for computing sub-graph

isomorphism. This architecture can be implemented efficiently in VLSI. Graph isomorphism is well known to belong to the computationally difficult class of problems known as the NP-complete problems.

Section 11 contains a large study by Hughes Research Laboratories on the suitability of a variety of parallel processing architectures, current and proposed. The study is necessarily qualitative in nature as many of these systems are not completely defined and there is also a lack of consensus on optimal IU algorithms for some tasks. Nonetheless, this study should be a valuable guide in the design of future architectures for IU.

A special purpose device called RADIUS was constructed by the Hughes Research Laboratories (under a sub-contract). This system uses novel residue arithmetic operations to allow a variety of arithmetic operations on images in real-time. The operation of most interest has been convolution. The kernel size of the current implementation is 5 x 5, but the design is modular and larger kernels can be obtained by replication. Details of this project have been described previously in [1].

REFERENCES

[1] R. Nevatia (Editor), Image Understanding Research Technical progress report,University of Southern California report #ISG102, October 1982.

[2] A. Huertas and R. Nevatia, "Detection of Buildings in Aerial Images Using Shape and Shadows," Proceedings of IJCAI-83, Karlsruhe, W. Germany, August 1983.

SECTION 2

VISUAL INSPECTION USING LINEAR FEATURES

Peter Kaufmann[1],
Gerard Medioni and Ramakant Nevatia

2.1 INTRODUCTION

Automatic inspection of printed circuit board assemblies is of obvious importance in electronics manufacturing. Several approaches to such inspection have been described in the past; a recent survey on automated visual inspection [1] contains over 200 references, including a large number for inspection of printed circuit boards and integrated circuits. We will not attempt a complete survey; basic techniques are described in textbooks on machine vision [2-3].

The various approaches to inspection can be characterized as belonging to the following classes. In one class, the image is compared to an ideal or model image directly, on a pixel-by-pixel basis, or by some form of area template matching. These methods are, of course, sensitive to variations in the lighting, reflectivity of the material and the size of the image. An alternative is to use a feature based description of the image and the model and to match the descriptions for inspection. Level and complexity of the description may vary; a typical example is the system described in [4-6]. Lastly, some methods attempt to find defects described generically, for example connecting wires to be of a certain minimum width or the corners to have certain characteristics [7-8]. These methods have the advantage of being applicable to a new part without changes, however, in many cases the defects to be inspected may be product dependent.

Our system is a feature based matching system. In our system, the symbolic descriptions to be matched are derived from line segments detected in an image. In addition, we also have a model of the expected defects. The part we have used in our tests is a printed circuit board used in digital watches in Switzerland. Its main components are:

- a printed circuit board, approximately 2cm by 2cm.
- a square, black plastic Integrated Chip with 8 soldering points
- an elongated, brass/ceramic capacitor with 2 soldering points
- a cylindrical metallic quartz connected to the board by 2 soldering points
- an elongated, brass battery contact riveted on the conductor.

The intensity of the parts of interest covers most of the black to white range and no effective threshold can be applied to the whole image to extract the desired parts. Figure 1 shows a complete printed circuit board with all elements in their proper locations. (All images shown in this paper were obtained by positioning the camera above the object,

---

[1]Peter Kaufmann was visiting USC and is now at E.T.A., Schild Strasse 17 Grenchen, Switzerland

the illumination was provided by two sources, a light table under the object and a light source normal to the table, and the resolution is 512 by 512 pixels).

The defects we wish to detect are:

-broken printed circuit board
-missing IC
-missing capacitor
-missing quartz
-battery contact out of place or missing

These defects are the ones that seemed to be of the major concern in the manufacturing process where this board is used. Here, we have not attempted inspection of the conductor paths; apparently, for simple boards, visual inspection of these paths is not of major concern.

2.2 DESCRIPTION OF THE MODEL

The model description consists of a set of line segments which are logically divided into submodels. Each submodel contains one or several segment chains which represent contours or outlines of parts or assemblies. Each of these submodels also has a descriptor which tells the comparison program how to interpret matches with the segment chains contained in the submodels. It contains for example a value for each segment chain which indicates the amount of correspondence with the part required to consider the chain as matched. The exact format of each submodel descriptor is defined as:

1. Lexical description of submodel (e.g. name of corresponding part in assembly).
2. Minimum required length of correspondence for each segment chain.
3. Maximum acceptable length of non-correspondence for each segment chain. (Note that the absence of some parts, such as the capacitor or integrated circuit, is indicated by too many line segments being visible, rather than not enough).
4. Message to be printed out in case of success or failure.

Figure 2 shows the outlines of parts of interest that constitute the segment component of the model.

2.3 SEGMENT ENCODING OF THE PART

To find the segment representation of an image, several processing steps have to be executed in sequence:

- Edge detection (followed by edge thinning and thresholding)
- Approximation of edges by line segments

These steps are only sketched here, a more complete description of the algorithms can be found in [9]. In this method, edge templates in six directions are convolved with the image and the direction with the highest output determines the magnitude and direction of the edge associated with a pixel. Based on direction and magnitude information, an edge thresholding and thinning operation follows. The threshold value is kept rather

low, so that no "good" edges are discarded at this early stage. Figure 3 shows the output of the edge detector using six 5x5 masks.

Next, the edges are linked to form continuous curves, and these curves are approximated by piece-wise linear segments (using the method described in [9]). Each line segment is described as follows:

- segment number (each segment has a unique identification)
- family number (unique for each linked segment chain)
- predecessor number (if the segment belongs to a chain)
- successor number (if the segment belongs to a chain)
- begin and end point coordinates
- strength (average contrast along the segment)
- length
- orientation angle

Although length and orientation are redundant, they are stored so that they are calculated only once. The algorithm to fit the line segments into the edge points is defined so that all segment end points are actual edge points and the normal distance between the line segment and the edge is never greater than a fixed number of pixels, say d. While generating these line segments, several filtering operations can be performed. Weak edges can be suppressed using length and strength properties of the edges. Figure 4 shows the line segments computed from the edges in figure 3. A line fitting tolerance, d, of 2 pixels was used and isolated segments shorter than 10 pixels were discarded.

2.4 COMPARING THE PART WITH THE MODEL

Now that we have the same representation for model and part under inspection, the next step is to find out how the part compares with the model. In order to compare them, we will assume identical position, orientation and size of model and part. If these constraints are not given by the layout of the inspection (fixtures, positioning device), then these parameters have to be roughly estimated in a preprocessing step which is discussed in section 4.1. Once we have model and part in an identical coordinate frame, corresponding segment chains have to be identified, as described in section 4.2.

2.4.1 Alignment of Model and Part

Given our data, which is a printed circuit board to be used in digital watches, we extract the chain of segments containing the longest vertical segment and match it with the outline of the left side of the model using a brute force technique, i.e. evaluating the quality of every possible set of matches, to estimate the parameters of the rotation and translation. This technique is effective because of the small number of elements involved in the matching, typically less than 5.

2.4.2 Matching the Part with the Model

The method used to match each part with the model is a variation of the Kernel method[10,11] successfully applied at USC on aerial images. The main differences stem from the fact that the structures to be matched were already aligned in a previous step. We will sketch here the main components of the process.

- Definitions

  We will denote the segments of the part as a_i, $1 \leq i \leq n$ and call them objects. We will denote the segments of the model as l_j, $1 \leq j \leq m$ and call them labels. The set $A=\{a_i\}$ is the scene.

  The set $L=\{l_j\}$ is the model.

  We want to compute the possibility $p(i,j)$ for object a_i to have label l_j. $p(i,j)$ can be either 0 or 1.

  The method presented here relies mostly on geometrical constraints, meaning that when we assign label l_j to object a_i, we expect to find an object a_h with the label l_k in a certain area dependent on (i,j,k). This area is noted w(i,j,k) and call the window w(i,j,k).

  Finally, we define the relation **C**, "is compatible with", between (i,j) and (h,k) as

  (i,j) IS COMPATIBLE WITH (j,k)
  <=> (i,j) **C** (h,k) <=> a_h in w(i,j,k) AND a_i in w(h,k,j).

  We need to check both predicates because the relation "is in w" is not symmetric.

  The method then proceeds as follows: Find a few pairs of very likely matches that are also mutually compatible, call these pairs the kernel and check each possible assignment for compatibility with this kernel.

- Assignment of possibilities

  Since scene and model are approximately aligned, this step is very simple: We will assign label l_j to object a_i if the corresponding segments have approximately the same position and orientation. The tolerance on orientation is large for small segments and very small for long segments. As we assign these labels, we mark every pair that is very closely matched as a good candidate for kernel element. We then extract the largest set of mutually compatible assignments from these candidates and call it the kernel. Then each other assignment is verified against this kernel for compatibility. Finally, we bridge accidental gaps in segment chains by looking at the labels assigned to the predecessor and successor of a given segment s; if both have a label, then we globally match chains and force a label on s.

- Interpretation

  Once scene and model have been matched, we generate a description of this match with each submodel part based on the length of matched chains. Each submodel contains the minimum and maximum bounds required for acceptance.

Results of this matching process are shown and discussed in the next section.

2.5 RESULTS AND CONCLUSIONS

Figures 5 to 10 show the results obtained with 6 different instances of the given printed circuit board representing different possible configurations. For each example, figure (a) shows the segments matched with the model of figure 2 superimposed on the image of the part and figure (b) gives a detailed explanation of the matching evaluation. For each sub-part of interest, the program outputs whether that sub-part is found and if so how much of the total length of model segments has been matched - this can be a measure of the quality of the match. A defect can be indicated by not enough of the model segments being found, or by too much of the segment length being visible. The acceptable limits are also given in the result figures.

It is clear that our program is able to detect the desired defects for the examples shown. However, the matching system would only be a component of an inspection system. There is much additional useful information in the images that we have not utilized. For example, we indicated the presence of the quartz crystal merely by the boundaries of the hole being invisible. Many lines, parallel to the axis of the quartz, are also typically detected by the line finder, and these along with some use of intensity variations could be used to more accurately verify that the correct part was in place (e.g. as in [12]). Thus, our system may be regarded as a top-level cueing mechanism that detects major defects to be verified by a more specific, goal-oriented inspection module.

REFERENCES

[1] R. T. Chin and C. A. Harlow, "Automated Visual Inspection: A Survey," IEEE Trans. PAMI, Vol. 11, No. 6, November 1982, pp. 557-573.

[2] R. Nevatia, Machine Perception, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[3] D. Ballard and C Brown, Computer Vision, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[4] M. L. Baird, "SIGHT-1: A Computer Vision System for Automated IC Chip Manufacture," IEEE Trans. on SMC, Vol. 8, No. 2, February 1978, pp. 133-139.

[5] W. A. Perkins, "A Model Based Vision System for Industrial Parts," IEEE Transactions on Computers, Vol. C27, February 1978, pp. 126-143.

[6] G. L. Agin, "Computer Vision Systems for Industrial Inspection and Assembly," Computer, Vol. 13, No. 5, May 1980, pp. 11-20.

[7] M. Ejiri, T. Uno, M. Mese and S. Ikeda, "A Process for Detecting Defects in Complicated Patterns," Computer Graphics and Image Processing, Vol. 2, 1973, pp. 326-339.

[8] J. R. Jarvis, "Visual Inspection Automation," Computer, Vol. 13, No. 5, May 1980, pp. 32-38.

[9] R. Nevatia and K. R. Babu, "Linear Feature Extraction and Description," Computer Graphics And Image Processing, Vol. 13, July 1980, pp. 257-269.

[10] G. G. Medioni, "Matching Linear Features of Images and Maps," Proceedings of the ARPA Image Understanding Workshop, Palo Alto, California, September 1982, pp. 103-111.

[11] G. G. Medioni, "Matching High Level Features of an Aerial Image with a Map or Another Image," Proceedings of Workshop on Computer Vision, Rindge, N.H., August 1982, pp. 113-115.

[12] B. K. P. Horn, "A Problem in Computer Vision: Orienting Silicon Integrated Circuits Chips for Lead Bonding," Computer Graphics and Image Processing, Vol. 4, No. 3, September 1975, pp. 294-303.

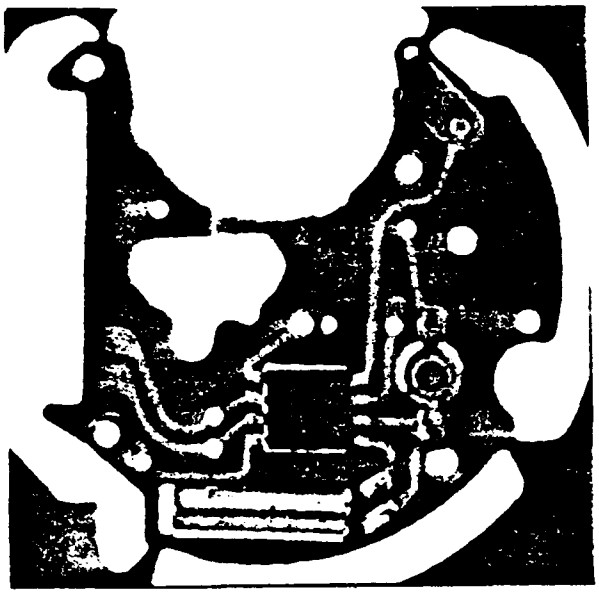


Figure 1: Complete Printed Circuit Board

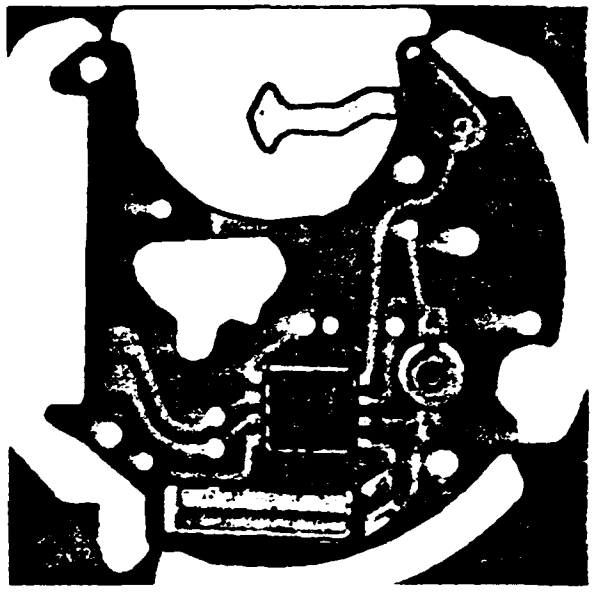


Figure 2: Outline of the model

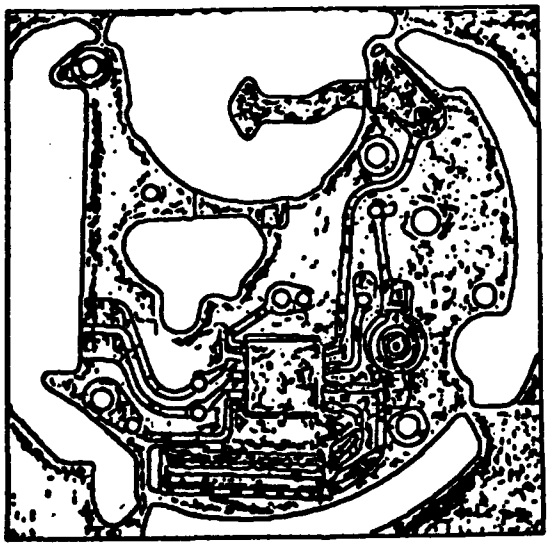


Figure 3: Edges using 5x5 masks

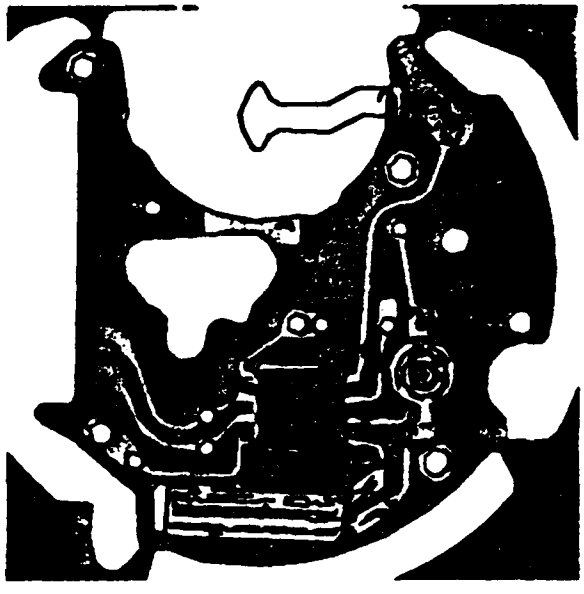
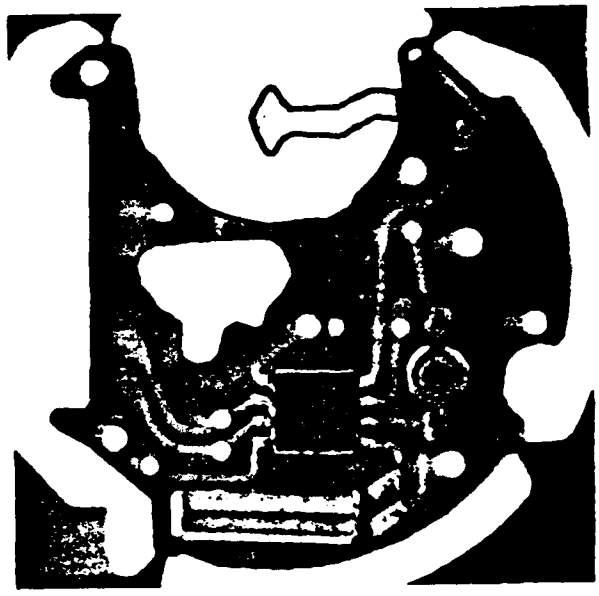


Figure 4: Line segments

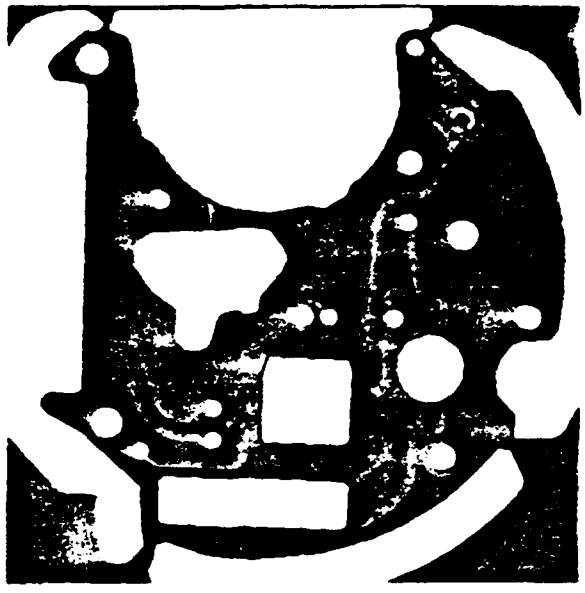


(a)

```
the part:  outline/top_right_corner is found
   not matched: .0000000        limit: 9.016455
   matched:     60.10970        limit: 51.09325
   model length: 60.10970
the part:  outline/left_side is found
   not matched: .0000000        limit: 86.02527
   matched:     573.5018        limit: 487.4765
   model length: 573.5018
the part:  outline/top is found
   not matched: .0000000        limit: 78.13128
   matched:     390.6564        limit: 312.5251
   model length: 390.6564
the part:  outline/right_side is found
   not matched: .0000000        limit: 65.95770
   matched:     439.7180        limit: 373.7603
   model length: 439.7180
the part:  outline/bottom is found
   not matched: .0000000        limit: 50.38727
   matched:     335.9151        limit: 285.5279
   model length: 335.9151
the part:  center_hole is found
   not matched: .0000000        limit: 62.00780
   matched:     413.3853        limit: 351.3775
   model length: 413.3853
the part:  capacitor is found
   not matched: 14.00000        limit: 9.855159
   matched:     162.4427        limit: 187.2480
   model length: 197.1032
the part:  Integrated Circuit is in place
   not matched: 81.78885        limit: 15.12139
   matched:     163.3025        limit: 287.3064
   model length: 302.4278
the part:  Quartz is in place
   not matched: 148.2147        limit: 20.80467
   matched:     167.5798        limit: 395.2887
   model length: 416.0933
the part:  Battery Contact is found
   not matched: .0000000        limit: 48.45157
   matched:     359.0243        limit: 274.5589
   model length: 323.0105
```

(b)

Figure 5: Matching results

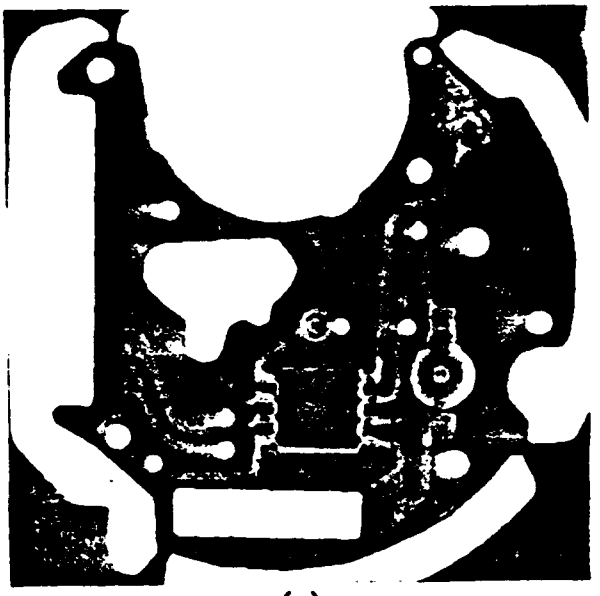


(a)

```
the part:  outline/top_right_corner is found
   not matched: .0000000        limit: 9.016455
   matched:     75.83074        limit: 51.09325
   model length: 60.10970
the part:  outline/left_side is found
   not matched: .0000000        limit: 86.02527
   matched:     574.5556        limit: 487.4765
   model length: 573.5018
the part:  outline/top is found
   not matched: 67.23146        limit: 78.13128
   matched:     326.8160        limit: 312.5251
   model length: 390.6564
the part:  outline/right_side is found
   not matched: .0000000        limit: 65.95770
   matched:     440.0282        limit: 373.7603
   model length: 439.7180
the part:  outline/bottom is found
   not matched: .0000000        limit: 50.38727
   matched:     337.7853        limit: 285.5279
   model length: 335.9151
the part:  center_hole is found
   not matched: .0000000        limit: 62.00780
   matched:     414.1680        limit: 351.3775
   model length: 413.3853
the part:  capacitor is missing or not in place
   not matched: .0000000        limit: 9.855159
   matched:     197.1032        limit: 187.2480
   model length: 197.1032
the part:  Integrated Circuit is missing or not in place
   not matched: .0000000        limit: 15.12139
   matched:     302.4278        limit: 287.3064
   model length: 302.4278
the part:  Quartz is missing or not in place
   not matched: .0000000        limit: 20.80467
   matched:     416.0933        limit: 395.2887
   model length: 416.0933
the part:  Battery Contact is not found
   not matched: 323.0105        limit: 48.45157
   matched:     .0000000        limit: 274.5589
   model length: 323.0105
```

(b)

Figure 6: Matching results

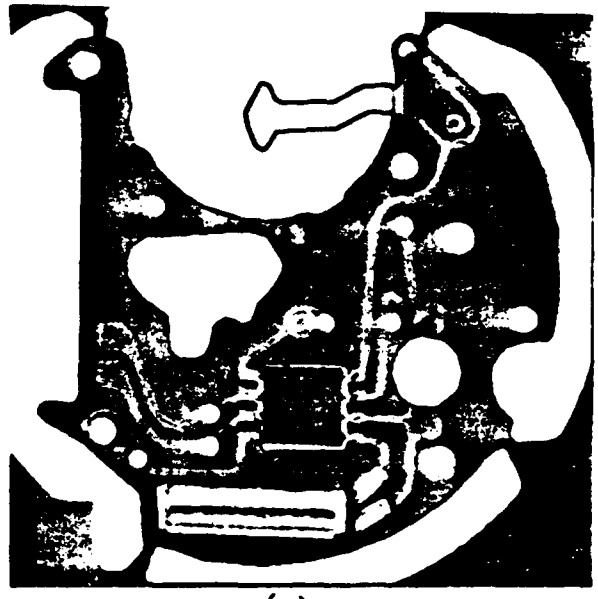


(a)

```
the part:  outline/top_right_corner is found
   not matched: .0000000        limit: 9.016455
   matched:     69.96133        limit: 51.09325
   model length: 60.10970
the part:  outline/left_side is found
   not matched: .0000000        limit: 86.02527
   matched:     604.6619        limit: 487.4765
   model length: 573.5018
the part:  outline/top is found
   not matched: .0000000        limit: 78.13128
   matched:     402.0784        limit: 312.5251
   model length: 390.6564
the part:  outline/right_side is found
   not matched: .0000000        limit: 65.95770
   matched:     444.0939        limit: 373.7603
   model length: 439.7180
the part:  outline/bottom is found
   not matched: .0000000        limit: 50.38727
   matched:     333.7796        limit: 285.5279
   model length: 335.9151
the part:  center_hole is found
   not matched: .0000000        limit: 62.00780
   matched:     413.3684        limit: 351.3775
   model length: 413.3853
the part:  capacitor is found
   not matched: 14.03567        limit: 9.855159
   matched:     149.6836        limit: 187.2480
   model length: 197.1032
the part:  Integrated Circuit is in place
   not matched: 10.29563        limit: 15.12139
   matched:     194.5388        limit: 287.3064
   model length: 302.4278
the part:  Quartz is missing or not in place
   not matched: .0000000        limit: 20.80467
   matched:     414.1096        limit: 395.2887
   model length: 416.0933
the part:  Battery Contact is not foud
   not matched: 323.0105        limit: 48.45157
   matched:     .0000000        limit: 274.5589
   model length: 323.0105
```

(b)

Figure 7: Matching results

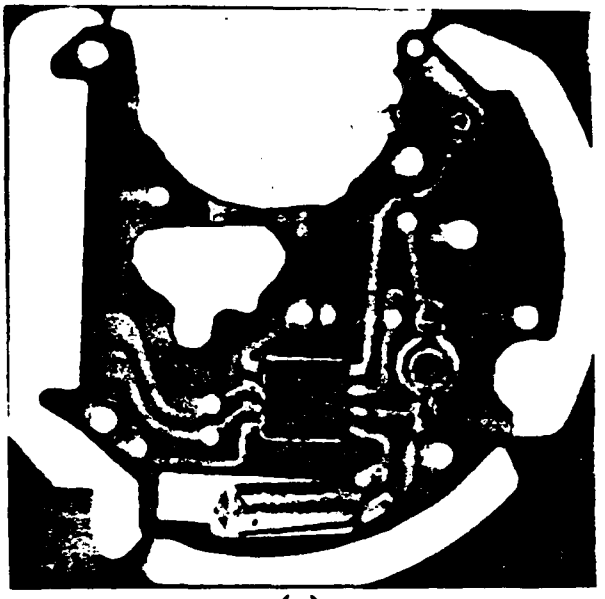


(a)

```
the part:  outline/top_right_corner is found
   not matched: .0000000        limit: 9.016455
   matched:     59.29365        limit: 51.09325
   model length: 60.10970
the part:  outline/left_side is found
   not matched: .0000000        limit: 86.02527
   matched:     581.3698        limit: 487.4765
   model length: 573.5018
the part:  outline/top is found
   not matched: .0000000        limit: 78.13128
   matched:     414.3125        limit: 312.5251
   model length: 390.6564
the part:  outline/right_side is found
   not matched: .0000000        limit: 65.95770
   matched:     450.9297        limit: 373.7603
   model length: 439.7180
the part:  outline/bottom is found
   not matched: .0000000        limit: 50.38727
   matched:     330.0233        limit: 285.5279
   model length: 335.9151
the part:  center_hole is found
   not matched: .0000000        limit: 62.00780
   matched:     415.7958        limit: 351.3775
   model length: 413.3853
the part:  capacitor is missing or not in place
   not matched: .0000000        limit: 9.855159
   matched:     199.0475        limit: 187.2480
   model length: 197.1032
the part:  Integrated Circuit is in place
   not matched: .0000000        limit: 15.12139
   matched:     220.7356        limit: 287.3064
   model length: 302.4278
the part:  Quartz is in place
   not matched: .0000000        limit: 20.80467
   matched:     365.5972        limit: 395.2887
   model length: 416.0933
the part:  Battery Contact is found
   not matched: .0000000        limit: 48.45157
   matched:     339.1811        limit: 274.5589
   model length: 323.0105
```

(b)

Figure 8: Matching results

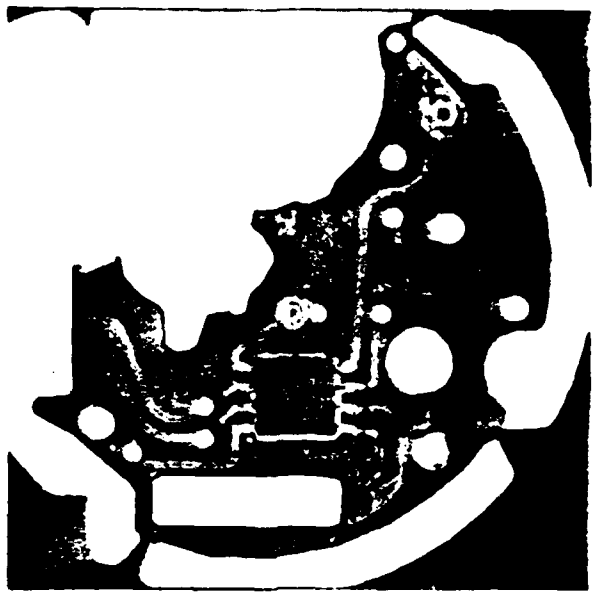


(a)

```
the part:  outline/top_right_corner is found
   not matched: .0000000        limit: 9.016455
   matched:     58.25959        limit: 51.09325
   model length: 60.10970
the part:  outline/left_side is found
   not matched: .0000000        limit: 86.02527
   matched:     571.9929        limit: 487.4765
   model length: 573.5018
the part:  outline/top is found
   not matched: 46.71318        limit: 78.13128
   matched:     356.0725        limit: 312.5251
   model length: 390.6564
the part:  outline/right_side is found
   not matched: .0000000        limit: 65.95770
   matched:     438.3224        limit: 373.7603
   model length: 439.7180
the part:  outline/bottom is found
   not matched: .0000000        limit: 50.38727
   matched:     368.0060        limit: 285.5279
   model length: 335.9151
the part:  center_hole is found
   not matched: .0000000        limit: 62.00780
   matched:     412.7569        limit: 351.3775
   model length: 413.3853
the part:  capacitor is found
   not matched: .0000000        limit: 9.855159
   matched:     164.6381        limit: 187.2480
   model length: 197.1032
the part:  Integrated Circuit is in place
   not matched: 73.30357        limit: 15.12139
   matched:     199.4696        limit: 287.3064
   model length: 302.4278
the part:  Quartz is in place
   not matched: 45.26370        limit: 20.80467
   matched:     333.6798        limit: 395.2887
   model length: 416.0933
the part:  Battery Contact is not found
   not matched: 323.0105        limit: 48.45157
   matched:     .0000000        limit: 274.5589
   model length: 323.0105
```

(b)

Figure 9: Matching results



(a)

```
the part:  outline/top_right_corner is found
   not matched: .0000000        limit: 9.016455
   matched:     55.81174        limit: 51.09325
   model length: 60.10970
the part:  outline/left_side is not found
   not matched: 102.8470        limit: 86.02527
   matched:     319.4996        limit: 487.4765
   model length: 573.5018
the part:  outline/top is not found
   not matched: 249.8139        limit: 78.13128
   matched:     125.5273        limit: 312.5251
   model length: 390.6564
the part:  outline/right_side is found
   not matched: .0000000        limit: 65.95770
   matched:     435.3755        limit: 373.7603
   model length: 439.7180
the part:  outline/bottom is found
   not matched: .0000000        limit: 50.38727
   matched:     366.6647        limit: 285.5279
   model length: 335.9151
the part:  center_hole is not found
   not matched: 114.4814        limit: 62.00780
   matched:     268.0096        limit: 351.3775
   model length: 413.3853
the part:  capacitor is missing or not in place
   not matched: .0000000        limit: 9.855159
   matched:     197.5216        limit: 187.2480
   model length: 197.1032
the part:  Integrated Circuit is in place
   not matched: .0000000        limit: 15.12139
   matched:     161.5724        limit: 287.3064
   model length: 302.4278
the part:  Quartz is missing or not in place
   not matched: .0000000        limit: 20.80467
   matched:     414.2710        limit: 395.2887
   model length: 416.0933
the part:  Battery Contact is not found
   not matched: 323.0105        limit: 48.45157
   matched:     .0000000        limit: 274.5589
   model length: 323.0105
```

(b)

Figure 10: Matching results

SECTION 3

RELAXATION MATCHING TECHNIQUES - A COMPARISON

Keith E. Price

3.1 INTRODUCTION

Matching of images and descriptions has many different uses and can be performed at several different levels. Some matching tasks require that very precise corresponding locations be computed (e.g., stereo depth computation, pixels level change detection). But for many tasks, matching at a higher level (i.e., finding correspondences between large areas) is best. This paper discusses results of using a variety of relaxation techniques in a general symbolic level image matching system applied to the task of matching an image and an a priori description of the scene (a model), and the task of matching two images to find the location of an object in two different views. Thus we use this program to find correspondences between areas of the images (or objects) rather than to find a pixel level mapping between them.

3.2 BACKGROUND

The work reported here represents an extension of earlier relaxation based symbolic matching efforts [1]. A variety of other image matching techniques have been developed for different tasks. Moravec [2] has developed a system which locates feature points in one image (essentially corners) and uses a correlation based matching procedure at multiple resolutions to efficiently find a set of corresponding points in the two images. This system is intended for land based robot navigation which uses the three dimensional information from these feature points for navigation. A stereo system developed by Baker [3] generates a complete disparity map starting from edge correspondences. The disparities can be used for depth computations if the camera positions are known. These two (and many other similar efforts) concentrate on precise matching of image data to obtain 3-dimensional descriptions.

Several systems which work on a variety of symbolic representations have also been developed. Barnard and Thompson [4] have developed a relaxation based motion analysis program which finds corresponding feature points in two images. The feature points are similar to those of Moravec [2], but they are located in both images. Wong et al. [5] also use a relaxation procedure to match corners which are detected in pairs of images. This system allows arbitrary translations and rotations of the camera. Clark et al. [6] have developed a system to match line like structures (generally either edges or region boundaries). The program uses three initial matching line pairs to get a mapping between the two images. The quality of the match depends on how well all the other lines match, and the best match is determined by trying all possible triples of matching lines. The number of possible triples is limited by the allowable transformations, i.e., given one match, the possible matches for the other two are very restricted. Gennery [7] extracts simple description of objects and uses a tree searching procedure to find the best match.

The primary relaxation procedure in this paper is developed more fully in [1, 8] and differs from other methods in its gradient optimization approach. The other alternative relaxation updating schemes used in the comparison are the basic method of Rosenfeld et al. [9], the product combination rule of Peleg [10], and the optimization method of Hummel and Zucker [11].

3.3 SYMBOLIC DESCRIPTION

This matching system uses feature-based symbolic descriptions for its input. The description of an idealized version of the scene (a model) is developed by the user through an interactive procedure. The image descriptions are derived automatically from the input images. The underlying descriptive mechanism is a semantic network. The nodes of the network are the basic objects with associated feature values and the links indicate the relations between objects.

The basic objects used in the image description are regions or linear features extracted by automatic segmentation procedures [12, 13]. These procedures produce a set of objects composed of connected regions which are homogeneous with respect to some feature in the input image [12] and long narrow objects which differ from the background on both sides and can be represented as sequence of straight line segments [13]. Only the important objects are described in the model. The automatic image segmentation produces many objects which are not included in the model (as many as 100-300 elements). The model description determines the outcome of the matching procedure and can also be used to guide the segmentation procedure [14].

The description is completed by extracting features of the regions and linear objects. The features are those which can be easily computed from the data and which are reasonably consistent. These features include average values of the image parameters (intensity, colors, etc.), size, location, texture, and simple shape measures (length to width ration, fraction of minimum bounding rectangle filled by the object, perimeter /area, etc.). Relations included in the description are also those which are easily computed; such as adjacency, relative position, (north of, east of, etc.), near by, and an explicit indication of not near by.

3.4 MATCHING

The basic goal for the matching procedure is to determine which elements in the image correspond to the given objects in the model. Most of the objects cannot be recognized by only their feature values. They require contextual information to be correctly located. An important idea used by the matching system is to locate a small set of corresponding objects using feature values and available contextual information. These initial islands of confidence provide the context needed for finding correspondences for the less well defined objects. Finally, when most objects are assigned, the matching can be done solely on the basis of context, thus radical differences in a few objects do not cause the matching program to fail.

The basic operation of the matching system is outlined in Fig. 1. In the large outer

loop a set of possible matching regions is determined for every element in the model. Each of these possible assignments has a rating (probability) based on how well the model and image elements correspond. These ratings are refined by the relaxation procedure in the inside loop, until one or more model elements have one highly likely assignment (usually a probability threshold of about 0.75 or 0.8). At this point a firm assignment is made and all likely assignments are recomputed using these assigned elements to give the context for the match. The inner relaxation procedure updates the probabilities of the assignment based on how compatible the assignment is with the assignments of its neighbors in the graph (i.e., objects linked by relations). We use a variety of relaxation schemes [1, 8, 9, 10, 11, 15] in this loop, with the criteria optimizing method in [1, 8] giving the best results.

The importance of this two level procedure is clear when an analysis of relaxation updating is made. Relaxation can be viewed as moving around in a multidimensional space searching for the global maximum of some function (such as overall match quality). But, the search is constrained to find a local maximum near the initial assignment [11]. The reinitialization step moves the search from the vicinity of one local maximum to another, which should be as high or higher.

3.4.1 Matching Details

The quality of match between two elements (one each from the model and image or from two different images) is given by the weighted sum of the magnitude of the feature value differences,

$$R(u, n) = \sum_{k=1}^{m} |V_{uk} - V_{nk}| \, W_k S_k \qquad (1)$$

where u is an element from the model n from the image, m is the number of features being considered, and $V_{uk}(V_{nk})$ is the value of the k^{th} feature of element u(n). W_k is a normalization weight (the same for all tasks) to equalize the impact from all features. S_k is the task dependent strength of a given feature. These strength values distinguish between important, average, and unimportant features. The ratio of the strength values is 5:1 and there is a fourth strength, zero, which indicates a feature is not used. This rating function is converted to the range [0, 1] by

$$f(u, n) = \frac{a}{R(u, n) + a} \qquad (2)$$

where a is a constant which controls how steep the differences function is. A value of 1 (a sharply declining function) produces the best results with the optimization updating approach. A larger value such as 10 should be used when using the product combination method [10]. Relations are treated like features in computing their contribution to the match rating. V_{uk} is the number of relations of type k which are specified in the model and V_{nk} is the number which actually occur in the image. Figure 2 illustrates how these values are computed for a given u_i. For each possible corresponding region n_k, check all u_j (in the model) which are related to u_i to see if the given correspondence (n_l) for u_j is properly related to n_k. When computing the initial probabilities of a match, only those u_j

which have been previously assigned can be considered.

The relaxation procedures require a function which measures the compatibility of a particular assignment n_k for u_i with the current possible assignments at all neighboring (related) units. This is defined by

$$Q_i(n_k) = \frac{1}{|N(u_i)|} \sum_{u_j \text{ in } N(u_i)} \sum_{n_l \text{ in } W_j} c(u_i, n_k, u_j, n_l) p_j(n_l)$$

$$+ \quad f(u_i, n_k) p_i(n_k) \tag{3}$$

Where N_i is the set of objects related to u_i, $|N_i|$ is the number of neighbors, is a factor between 0 and 1 that adjusts the relative importance of features versus relations (0.1 to 0.25 is the usual range), $p_i(n_k)$ is the current probability for assigning name n_k to unit u_i, W_j is the set of likely assignments of u_j (for efficiency and improved results we generally use only the one most likely assignment here). $c(u_i, n_k, u_j, n_l)$ is the same as $f(u_i, n_k)$ except that only relations between u_i and u_j are considered. The vector $\bar{Q}_i$ is used directly in the updating step without normalization, which simplifies the computation. The iterative updating is given by

$$\bar{p}_i^{(n+1)} = \bar{p}_i^{(n)} + \rho_n P_i\{\bar{g}_i^{(n)}\} \tag{4}$$

where ρ_n is a positive step size to control the convergence speed, P_i is a linear projection operator to maintain the constraint on $\bar{p}_i^{(n+1)}$ that it is a probability vector, and $\bar{g}_i^{(n)}$ is an explicit gradient function determined by the optimization criteria [1, 8].

$$g_i(n_k) = -Q_i(n_k) - p_i(n_k) \quad f(u_i, n_k) \tag{5}$$

$$- \sum_{u_j \text{ s.t. } u_i \in N(u_j)} \frac{1}{|N(u_j)|} \sum_{n_l \text{ in } W_j} c(u_j, n_l, u_i, n_k) p_j(n_l)$$

Briefly, the gradient gives the direction of greatest change in the criteria and the updating function takes a step in this direction. An alternative formulation is the one by Hummel and Zucker [11]. They optimized a different function and thus do not compute the gradient, but, under the assumptions which apply here, the final updating step is the same except that $\bar{Q}_i$ is used rather than $\bar{g}_i$. The original method of Rosenfeld et al. [9] uses the same computation method for the compatibility measure, but has a different updating function.

$$p_i^{(n+1)}(n_k) = \frac{p_i^{(n)}(n_k) Q_i^{(n)}(n_k)}{\sum_{n_l \text{ in } N} p_i^{(n)}(n_l) Q_i^{(n)}(n_l)} \tag{6}$$

The product method of Peleg[10] uses the updating function given in Eq. 6, but combines the compatibility values using a product rather than a sum. This changes the outer summation in Eq. 3 to a product. If this product combining rule is used, then the constant (a) in Eq. 2 must be a much larger value (10 or 100) so that the match ratings are not all almost zero.

3.5 RESULTS

We have applied this system to a variety of images (generally two views of each scene, see Fig. 3, 4). For different views of the same scene, we use the same model. The results are presented as overlays on the original images, showing the border of regions or center lines of linear features. The labels are taken from the name given in the model, either the user derived model or the image which serves as a model. Table 1 summarizes the results. The results given here reflect the use of a post-match error elimination heuristic which eliminates errors based on ambiguous matches.

Figure 5 shows the results of matching the model to the first image of San Francisco area (Fig. 3). The errors in the second view are caused by the segmentation errors. The two sections of the Bay Bridge are missed by the linear feature extraction program and thus both cannot be matched correctly plus the island which is adjacent to the bridges and both portions of the bay is mismatched. (Note that the two sections of the bay were intended in the model description to be split by the bridges.) See Table 1 for a summary of the results.

Figure 6 gives the results for a subwindow of the low altitude aerial images (Fig. 4). The regions were extracted using a model based segmentation technique [14]. Different objects are segmented poorly in the two views, but the matching still works well for both. An alternative segmentation produced similar matching results (using the same model), even with differences in some of the extracted regions.

The two methods [1] and [11] based on an optimization approach give more correct matches with few errors. The product combination rule [10] converges very rapidly (as expected), but does not make the "difficult" assignments which require more iterations to incorporate the global context. The original method [9] was more uneven in its performance (some good, some poor). The product rule is the fastest - becaue of the product in the combinations, a few low match ratings quickly cause the overall rating of an assignment to approach zero. Because the number of iterations is greater, the original method takes longer than the product method. The two optimization approaches take even longer due to the increased number of iterations and the increased complexity of each step. Interestingly, the simpler method of Hummel and Zucker requires more time and, usually, a greater number of iterations, because fewer potential matches are eliminated (forced to a probability of zero) on each iteration.

Figures 7-9 present the image to image matching process. In Fig. 7 the first view is used as the model, and the second is used in Fig. 8 (the image used as the model is the one on the left). Figure 9 shows those pairs which occur in both cases. Table 2 gives the computed disparities for each of these 37 matched objects. The same general

comments apply here as in the preceeding paragraph. Note (see the summary in Table 1) that the product rule combination forces probabilities quickly to one or zero so that global context may be missed.

3.6 SUMMARY AND CONCLUSIONS

This paper compared the use of four different relaxation updating schemes with the same general matching system. The most consistent performance was the gradient based optimization approach detailed in [1]. There is a higher cost, in terms of the complexity of the necessary operations as reflected in the computation time. These results indicate that rapid convergence is not always an advantage, when the relaxation converges before enough global context has been included.

REFERENCES

[1] O. Faugeras and K. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," IEEE T-PAMI, Vol. 3, No. 6, Nov. 1981, pp. 638-642.

[2] H. Moravec, "Rover Visual Obstacle Avoidance," Proc. 7-IJCAI, Vancouver, B.C., Canada, Aug. 1981, pp. 785-790.

[3] H. Baker and T. Binford, "Depth from Edge and Intensity Based Stereo," Proc. 7-IJCAI, Vancouver, B.C., Canada, Aug. 1981, pp. 631-636.

[4] S. Barnard and W. Thompson, "Disparity Analysis of Images," IEEE T-PAMI, Vol. 2. No. 4, July 1980, pp. 333-340.

[5] C. Wong, H. Sun, S. Yada, and A. Rosenfeld, "Some Experiments in Relaxation Image Matching Using Corner Features," Univ. of Maryland, Computer Vision lab, Computer Science Center, TR-1071, 1981.

[6] C. Clark, D. Conti, W. Eckhardt, T. McCulloh, R. Nevatia, and D. Tseng, "Matching of Natural Terrain Scenes," in 5-ICPR, Miami, Fla., Dec. 1980, pp. 217-222.

[7] D. Gennery, "A Feature Based Scene Matcher," in 7-IJCAI, Vancouver, B.C., Canada, Aug. 1981, pp. 667-673.

[8] O. Faugeras and M. Berthod, "Improving Consistency and Reducing Ambiguity in Stochastic Labeling: An Optimization Approach," IEEE T-PAMI, Vol. 3, July 1981, pp. 412-424.

[9] A. Rosenfeld, R. Hummel, and S. Zucker, "Scene Labeling by Relaxation Operations," IEEE T-SMC, Vol. 6, June 1976, pp. 420-453.

[10] S. Peleg, "A New Probabilistic Relaxation Scheme," IEEE T-PAMI, Vol. 2, No. 4, July 1980, pp. 362-369.

[11] R. Hummel and S. Zucker, "On the Foundations of Relaxation Labeling Processes," McGill Computer Science TR-80-7, Montreal, Quebec, 1980.

[12] R. Ohlander, K. Price, and R. Reddy, "Picture Segmentation Using a Recursive Region Splitting Method," Comp. Graphics and Image Proc., Vol. 8, pp. 313-333, 1978.

[13] R. Nevatia and K. Babu, "Linear Feature Extraction and Description," Comp. Graphics and Image Proc., Vol. 13, 1980, pp. 257-269.

[14] K. Price, and G. Medioni, "Segmentation Using Scene Models," to be published.

[15] L. Kitchen, "Relaxation Applied to Matching Quantitative Relational Structures," IEEE T-SMC, Vol. 10, Feb. 1980, pp. 96-101.

[16] A. Hanson and E. Riseman, "VISIONS: a Computer System for Interpreting Scenes," in Computer Vision Systems, A. Hanson and E. Riseman (eds.), Academic Press, New York, 1978, pp. 303-333.

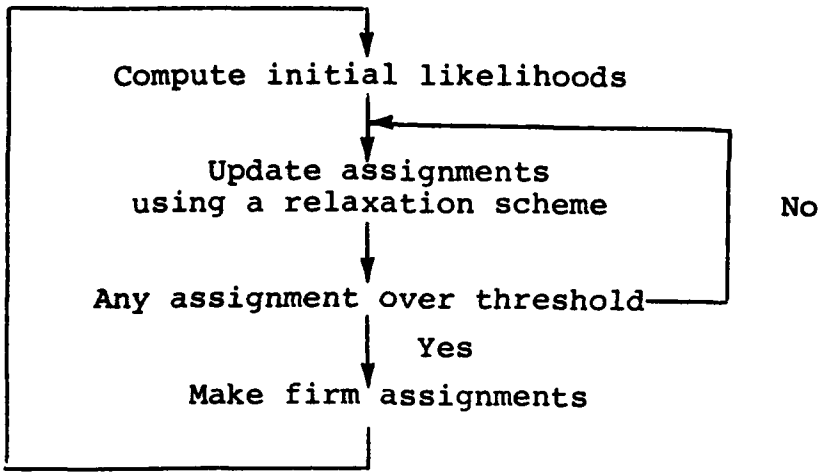


Figure 1: Basic operation of the matching system. Several different relaxation procedures can be used in the inner loop.

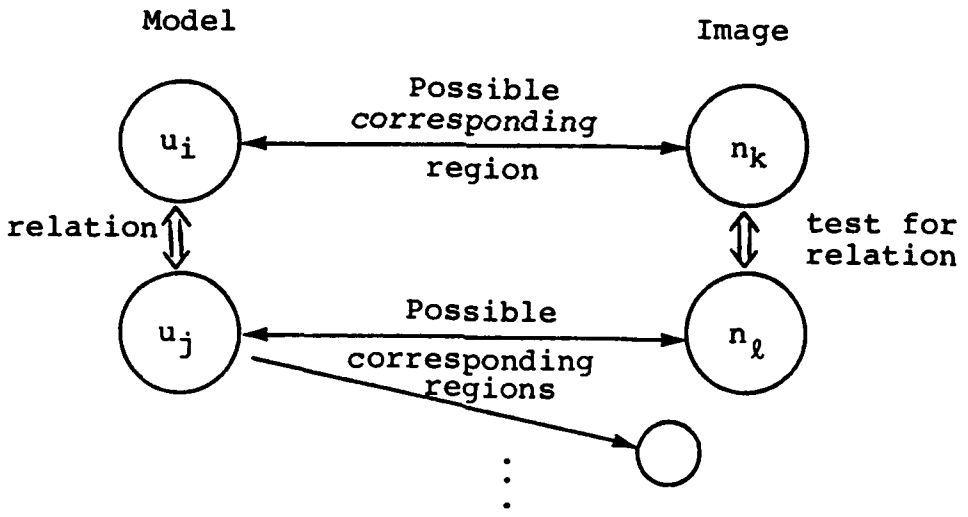
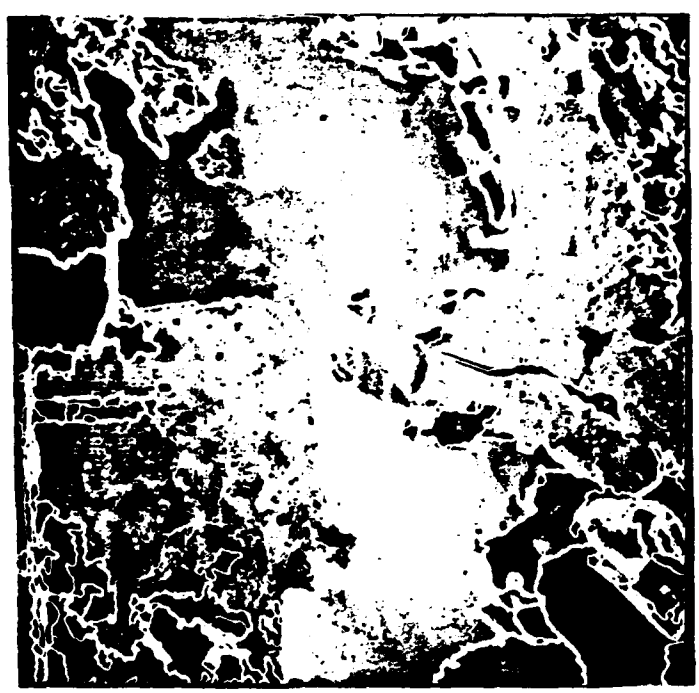


Figure 2: Computation of the match value for relations

(a)



(b)

Figure 3: Two high altitude aerial views of the San Francisco Bay area. With region based segmentations and linear features indicated.

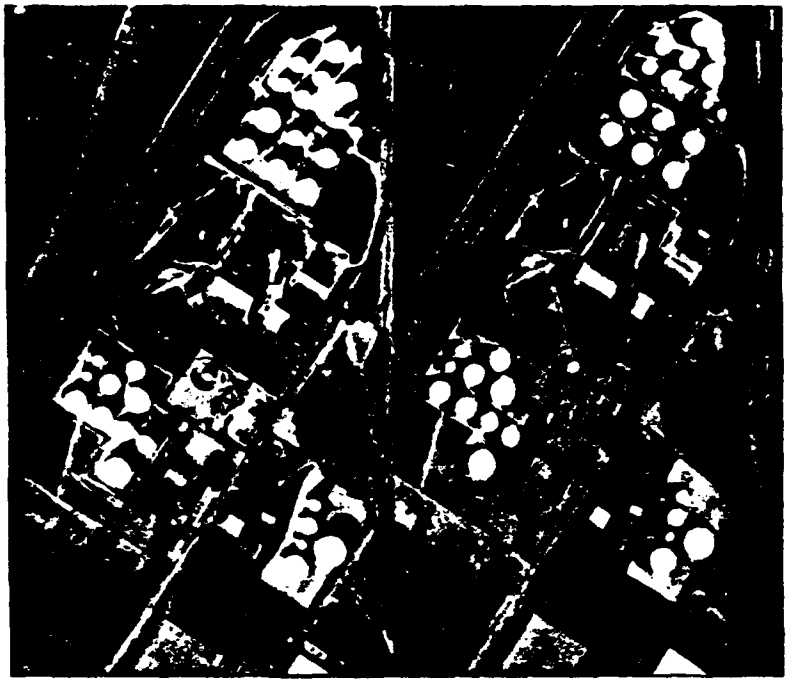


Figure 4: Two low altitude views of the Ft. Belvoir UA area

Figure 5: Matching results using optimization approach for the first San Francisco view

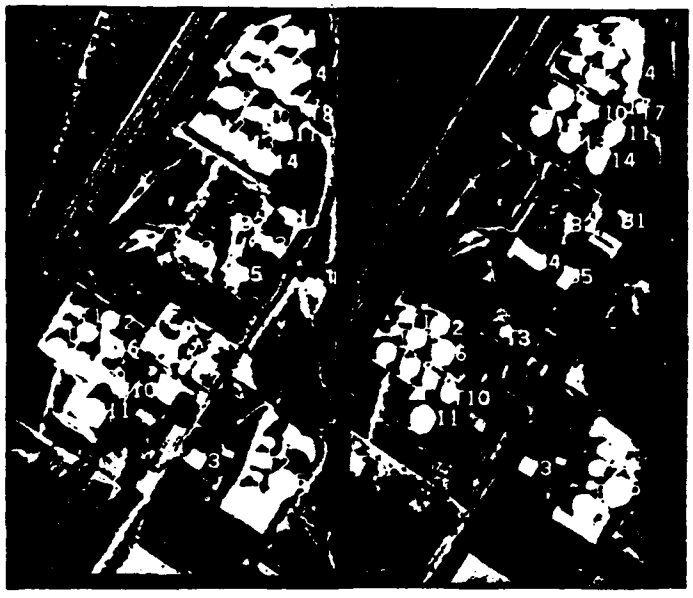


Figure 6: Matching (using the optimization approach) for 2 windows of the low altitude viewing with a scene model

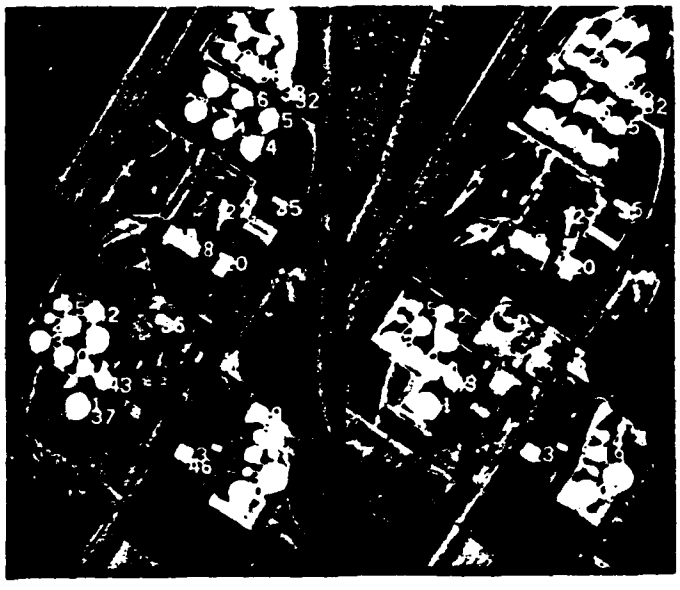


Figure 7: Image to image matching (using the optimization approach) for the two low altitude images. The image on the left served as the model.

Figure 8: Same as Fig. 7 except the other view is used as the model

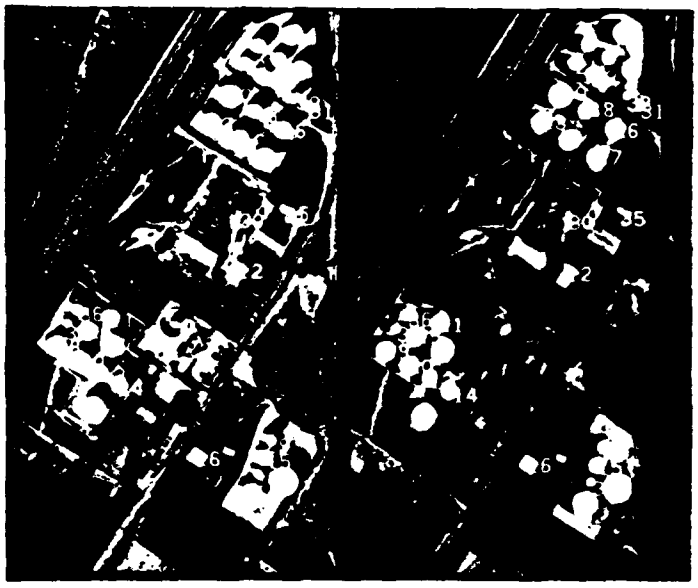


Figure 9: Continuation of results from Fig. 7 and 8.

SECTION 4

MATCHING CLOSED CONTOURS

Keith E. Price

4.1 INTRODUCTION

Closed boundary matching and line segment matching have been explored in a variety of contexts. Chow and Aggarwal [1] used contour matching for motion studies of simulated cloud patterns. Davis [2] used a relaxation based matching system for recognizing islands using their outlines. Bhanu [3] has looked at contour matching of occluded objects. Recently, Ayache [4] has given a method for accurately locating an object in a scene where there is substantial occlusion (or additional metal on the original object). Line segment matching (without considering closed contours) has been studied by Clark et al. [5] for arbitrary aerial views and Medioni [6] for stereo pairs.

The matching method here is an attempt to be more general in terms of the type of potential tasks and computationally simpler. The previous work has shown that corresponding segments in two views can be computed reliably and used for recognition or object matching. Also, it has shown that some correct corresponding segments can be located by simple, rotation invariant, features of the segments. The most important consideration for matching closed contours is that the order of segments in the two views must be the same (or in strictly reverse order if mirror images are allowed).

4.2 ALGORITHM DESCRIPTION

Rotation invariant features of line segments include the line segment length and the angle between consecutive segments. Using only these features, a given line segment in one view can readily match many segments in another view. But a consecutive sequence of border segments in one view should have few matches with consecutive or monotonically increasing sequences in the other view. Rather than comparing sequences of boundary segments, we will compute the potential matches and look for sequences.

4.2.1 Boundary Descriptions

The images are of a variety of small tools on a light table for a good contrast, but the clear handles of some of the tools do not always have a sufficient contrast between the objects and the background. The objects are extracted using a simple threshold to separate them from the bright background. The boundary of the region is transformed into a sequence of straight segments by a procedure originally developed for sequences of edge points [7]. The accuracy of the line segment representation is controlled by a parameter which gives the allowable deviation of the individual points from the straight line segments. Three representations are generally computed corresponding to a deviation of 1.2 (1 has an anomalous behavior), 2 and 4. Each boundary is represented as an ordered sequence of these line segments with length and orientation for each. Segments

are numbered in order around the boundary, thus consecutive segment numbers correspond to adjacent segments in the boundary. These segment sequences are called families. Each family corresponds to one region boundary with separate families for the interior boundaries of holes. Several families are included in the description for an image, with each family processed separately.

4.2.2 Initial Matching Segments

The matching procedure considers two families at a time, one from each of two images. Each segment in the first family is compared with each in the second family to determine if they can possibly match. If they do match, then the orientation difference is stored in an array (indexed by segment numbers), called a disparity array. Possible matches are determined by comparing the segment lengths and by comparing the angles between the current segments and their respective successors. Both of these tests use a threshold chosen by the user. The length threshold is a multiplicative factor greater than 1, thus we can use the test:

$$L(1)/t \leq L(2) \leq L(1)*t$$

where L(X) is the length of segment X and t is the threshold. At this point the length restriction is severe - around 1.3. The angle difference threshold depends on the resolution of the line segment representation, ranging from 90° for the lowest resolution version to 45° for the highest.

Each segment will have many possible matches using these two criteria, but there should be very few cases where several consecutive segments in one image match consecutive segments in the other all with similar orientation differences. Thus, we need to find long consecutive sequences of matching segments. As the first step, we find two matches, n with m and n+1 with m+1. These two matches are used as the starting point for a simple search to find a long sequence of matches where gaps in either sequence are allowed.

4.2.3 Initial Matching Sequence

From the pair of initial matches we search for the next (or previous by searching backwards) matching segments. We look at the diagonal segments next (increment both by one) then the points of the diagonal. In the following:

```
X . . . . .
. Y 3 8 5 .
. 2 1 6 3
. 7 5 4 1
. 4 2 0 9
```

X is the first point located and Y the second. The search starts at 1 then continues at 2,

3, etc, after 9 we continue at 0, 1, ... This search continues until another matching pair is located which has a disparity array value within the threshold range of the first pair. The same threshold is used as in the initial match computation. Here it is a limit on the difference between disparities, there is was used as a limit on the difference between the angles of segments and successors. Gaps between the last match and the new match are filled in as matchs when the new match occurs along the diagonal, i.e. both sequences jumping the same amount.

All possible sequences are located in the two families. If the longest sequence has enough points (5 for cases where good matches are expected, 3 as an extreme where nothing is known) then this set of matching segment pairs is used to determine the approximate transform to align the two families. The transform is one which would perfectly align a pair of matching segments. This pair is the one near the median, orientation difference (computed using the length of the segments as weights), which is longest and has both segments of the pairs near the same length. That is, starting at the median look for the longest segment where the ratio between the segment lengths (short/long) is greater than 0.8. The best two transformations are used for the transformation refinement (along with the best transformation from the second longest matching sequence - if it is close (0.7) to the length of the longest one). This transformation only applies for mapping between the pair of families, there will be many such transformations in the final complete match.

4.2.4 Transformation Refinement

Two (or possibly three) transformations for a pair of families have been generated which must be compared to select the best, for the final result. With a known transformation, we use different constraints for matching segments. A possible match is indicated if the segments overlap in position, or nearly overlap, and the orientations are similar - after the transformation has been applied to the appropriate segment.

Using each possible transformation, we compute the set of initial matching segments and the sequence of matching segments by the same procedures as above. A transformation is applied to all the segments in the first image and different features are used to determine a match. Position and orientation of the segments are used, with thresholds on the allowable differences of each. A disparity value (Euclidian distance) is stored in the disparity array and is used in the search for long sequences. Since a transformation is applied to segments in the first view, the disparities should all be near zero, but the matching ideas here are more general and can be used in a stereo problem where there is no angle transformation and disparity is the only way to separate various possibilities.

For each transformation we compute the likely matches and find the longest sequence. From these sequences we choose the longest to compute a final transformation. We use a disparity threshold of 10° and an orientation difference which varies from 90° for short segments to 20° for long segments. The search for long sequences allows wrap around matches - if one sequence hits the end of the sequence it can start over at the beginning while the second only increments by one.

4.2.5 Hierarchical Matching

Multiple resolution segment representations help improve speed and accuracy. The time for matching of two families depends on the number of segments in the two families, but the alignment is best when the segments very closely follow the contours of the object. We apply the two step matching procedure to the lowest resolution representation and obtain a set of matches for many of the families. At the next higher resolution we use the known transformation as the starting point and apply the transformation refinement operation twice. (The second step primarily finds which segments match with the updated transformation rather than a more accurate transform.) Families which had no match at the lower resolution are processed the same as at the lowest resolution - find initial matches using the length and angle with the successor, then refine the match using position and orientation.

4.2.6 Matching Summary

In summary, the matching procedure can be described as two passes of two processing steps applied to each pair of families.

Pass 1, step 1: Compute likely corresponding segments by comparing all segments with all others. Use segment length and the angle between a segment and its successor to determine the match.

Pass 1, step 2: Locate sequence of corresponding segments where the segment number increases monotonically in each image. Use these sequences to determine a good transformation to map one set into the other.

Pass 2, step 1: Using the transformation compute a new set of likely matching segments using segment position and orientation.

Pass 2, step 2: Locate sequences of monotonically increasing segments and determine a new transformation.

For multiple resolution data, Pass 2 is repeated as Pass 3 and 4, to determine corresponding segments at the higher resolution and yet a better transformation.

4.3 RESULTS

This matching program is intended to be somewhat general, it answers the two questions: Do these two sets of segments match? What transformation will align the view in the first image with the second? Because we wish the program to work even with occlusions, the program will indicate a good match when presented with two partially similar objects. If the task is recognition then an evaluation of the match quality would be necessary to determine which identification is best. In this paper, the results are for a basic matching task, not specifically recognition, but we do evaluate the matches and eliminate those which are much worse, based on number of matches, total disparity, total orientation differences, and total successor angle difference, than others

for the same family.

The input images are of a set of tools (two pairs of pliers, two small screwdrivers, one longer one with a similar handle, one large screwdriver and one short, fat one). A mechanical pencil and a fountain pen were also included. These last two had fewer segments in the representation and, in some cases appeared as mirror images and thus did not match as well. Two views of all nine objects, with no occlusions, were taken, plus two more views of a subset of the objects, and six other views with a variety of occlusions. The exact segment to segment match is not important since some segments only partially match, therefore, we will present the results as outlines taken from the first images transformed to line up with the objects in the second image.

Figure 1 shows the outline of the two images with all objects and no occlusions. These two images are matched with all the others (including with each other) in our experiments. Even though the images were digitized on a light table to obtain near perfect outlines, in some cases the clear handles of the screwdrivers cause problems. Figure 2-3 show some of the results - selected to show successes and problems.

4.3.1 Evaluation

The program locates most of the correct matches and many of the extra matches are with very similar objects. The differences between the two small screwdrivers are very minor and the handle of the long bladed screwdriver is almost the same as the two small ones, so these three often match all three possibilities (see Fig. 2). When two similar objects occlude each other, the match for both may be with the same sequence (see Fig. 3). Round objects can cause difficulties (even when there are small well defined "ears") since many different rotations will give a good match. We show no examples here, but we encountered this problem on an earlier similar data set and mention it as a known problem.

This matching procedure is reasonably efficient, with the total time depending on a number of factors - primarily the number of segments in the representation. For example, the matching for image 1 (Fig. 1a) with image 2 (Fig. 1b) (see Fig. 2 for the results) takes about 2 minutes 36 seconds. This includes matching at three resolutions for 9 objects in each image. Approximately 80% of the time is in computing the likely matches and 15% in searching for sequences of matches or computing the transformation. The times for the higher resolutions are not significantly greater than the lowest resolution because the matching is restricted to refining existing matches, not searching for new ones. The lowest resolution match uses 116 and 119 segments from the first and second view, respectively and compares 9 families in one view with all 9 in the second (i.e. test the match for 81 possible combinations). This requires a total of 47 seconds for all 81 comparisons. The implementation is on a PDP-10 with no special effort for low level efficiency.

4.4 CONCLUSIONS

We have proposed a simple relatively efficient matching procedure for comparing contours in scenes containing occlusions and multiple objects, which requires no iterative updating (relaxation). This procedure uses the order of segments around a boundary as the most important criterion for determining whether individual segments match. There are still some open problems, which are also problems for any other existing system. These include how to evaluate several different matches for use in a recognition system with a variety of similar objects. Another problem is the uniform use of holes which give multiple segment sequences for one region, and efficient handling of almost circular objects.

REFERENCES

[1] W.K. Chow and J.K. Aggarwal, "Computer Analysis of Planar Curvilinear Moving Images," IEEE - Trans. Computer Vol. 26, Feb. 1977, pp. 179-185.

[2] L. S. Davis, "Shape Matching Using Relaxation Techniques,"IEEE Trans. PAMI, Vol. 1, No. 1, Jan. 1979, pp. 60-72.

[3] B. Bhanu, "Recognition of Occluded Objects," Proc. IJCAI-83 Karlsruhe, W. Germany, Aug. 1983, pp. 1136-1138.

[4] N. J. Ayache, "A Model Based Vision System to Identify and Locate Partially Visible Industrial Parts," in Proc. IEEE Conf. on Computer Vision and Pattern Recognition, Arlington, VA, June 1983, pp. 492-494.

[5] C.S. Clark, D.K. Conti, W.O. Eckhardt, T.A. Mcculloh, R. Nevatia, and D.Y. Tseng, "Matching of Natural Terrain Scenes," in Proc. 5-ICPR, Miami, FL, Dec. 1980, pp. 217-222.

[6] G.G. Medioni and R. Nevatia, "Segment-Based Stereo Matching," Proc. DARPA IU Workshop, Arlington, VA, June 1983, pp. 128-136.

[7] R. Nevatia and K.R. Babu, "Linear Feature Extraction and Description," Comp. Graphics and Image Proc., Vol. 13, 1980, pp. 257-269.

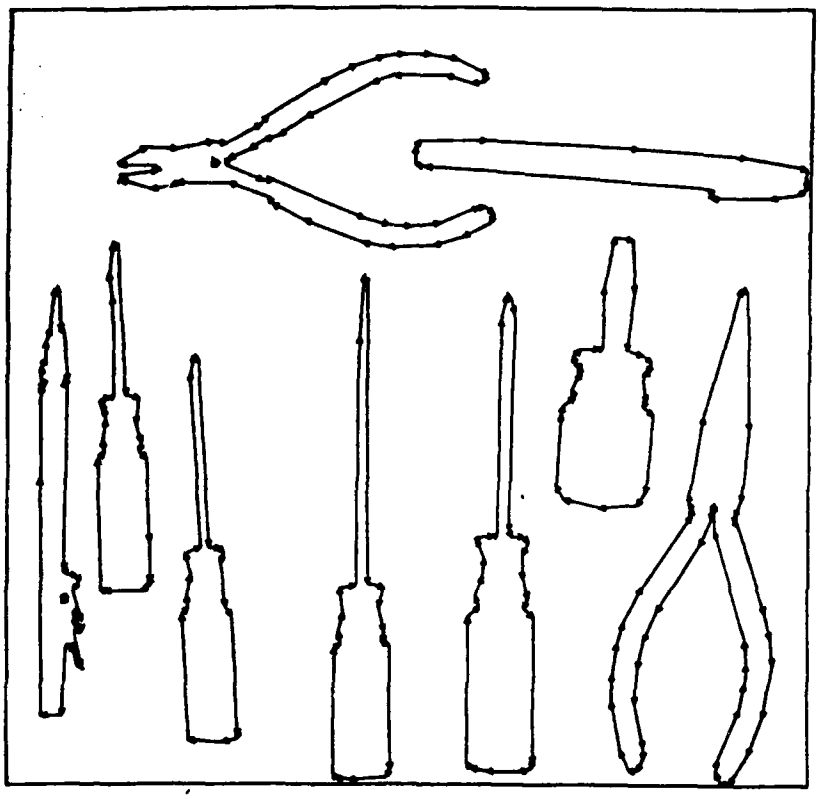


Figure 1a: Outlines of the first image

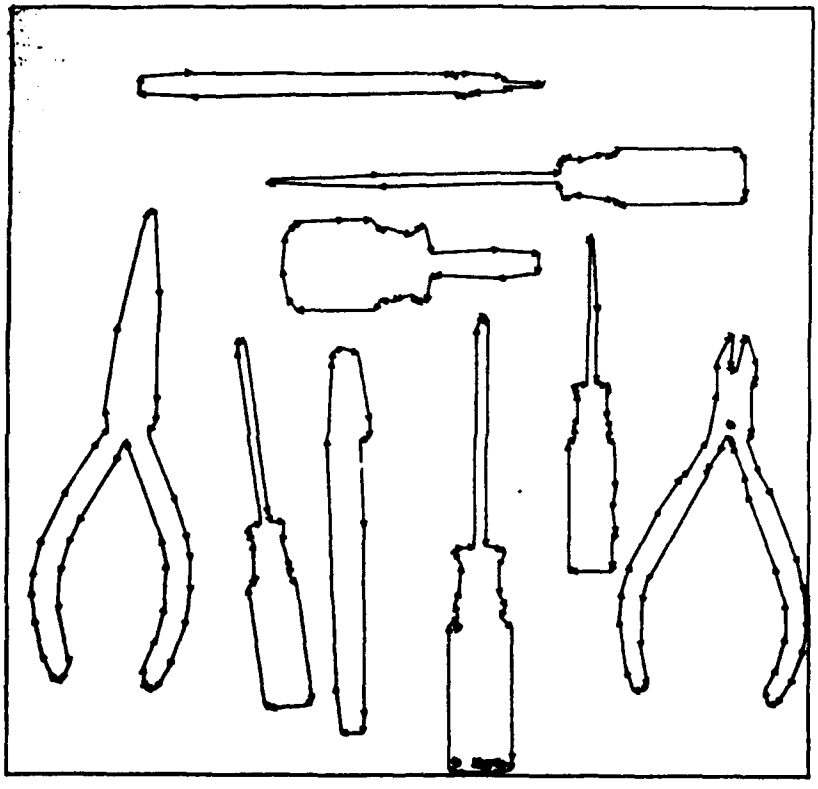


Figure 1b: Outlines of the objects in the second image

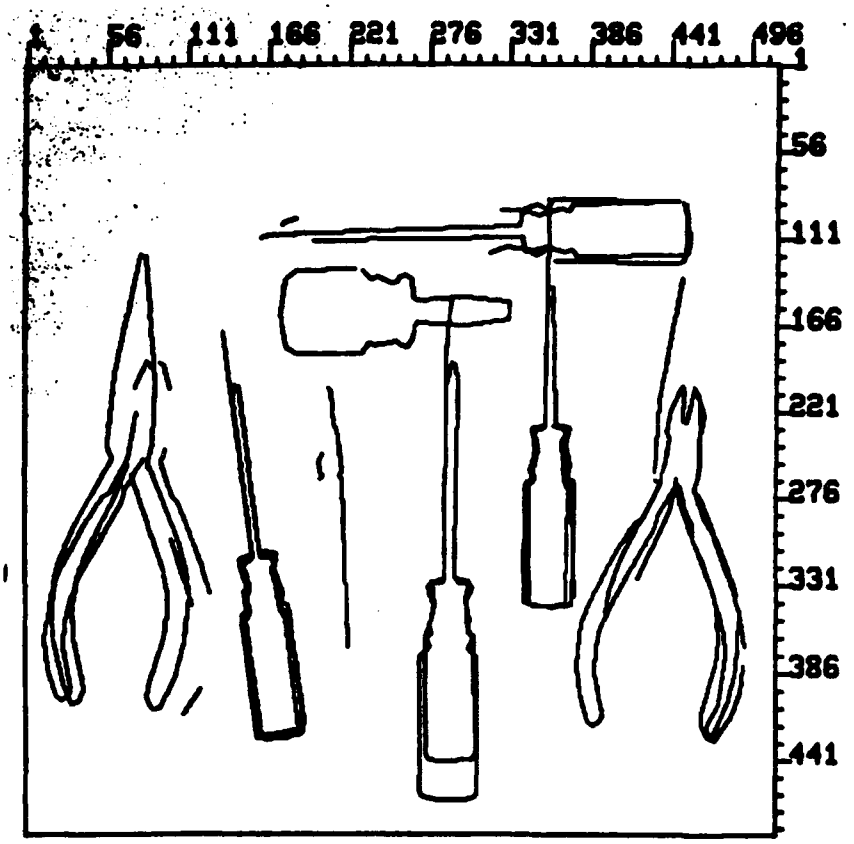


Figure 2: Matching of image 1 and image 2

The outlines of image 1 (Fig. 1a) are transformed to line up with the objects in image 2 (Fig. 1b) and displayed.

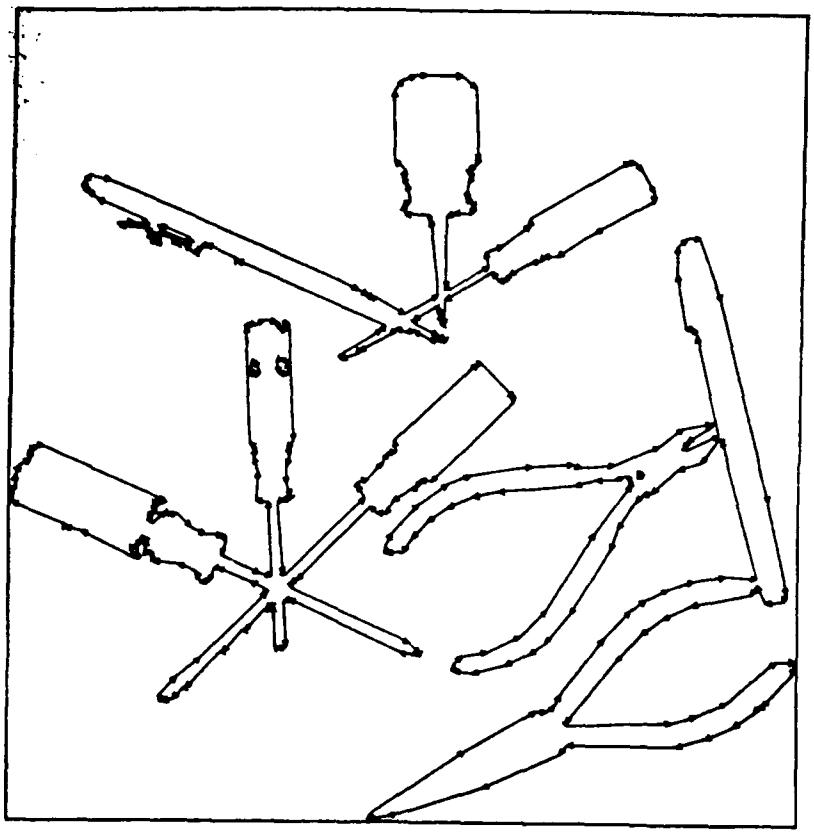


Figure 3a: Outlines from third image

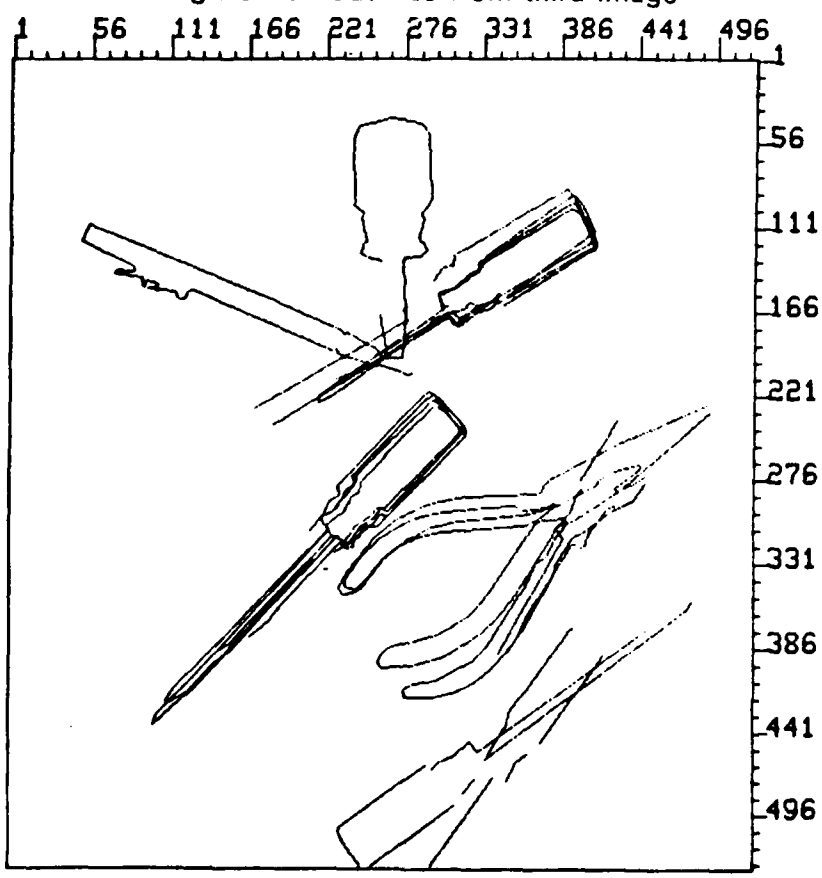


Figure 3b: Result of matching image 1 (Fig. 1a) with image 3 (Fig. 3a)

SECTION 5

SEGMENT-BASED STEREO MATCHING

Gerard G. Medioni and Ramakant Nevatia

5.1 INTRODUCTION

The human visual system perceives depth with no apparent effort and very few mistakes, but how it does so is not understood. Binocular stereopsis plays a key role in this process, and the straightforward extraction of depth it provides, once corresponding points are identified, makes it very attractive. Depth recovery is necessary in domains such as passive navigation [1, 2], cartography [3, 4], surveillance [5] and industrial robotics. Proposed solutions for the stereo problem follow a paradigm involving the following steps [6]:

-image acquisition,
-camera modeling,
-feature acquisition,
-image matching,
-depth determination,
-interpolation.

The hardest step is image matching, that is identifying corresponding points in two images, and this chapter is solely devoted to it. The next section reviews the existing systems that have been proposed so far, divided in two broad classes, area-based and edge-based, then we summarize our assumptions and give a formal description of the method. The fourth section presents results, and we then discuss extensions.

5.2 REVIEW OF EXISTING METHODS

Two classes of techniques have been used for stereo matching, area-based and feature-based.

5.2.1 *Area-based Stereo*

Ideally, one would like to find a corresponding pixel for each pixel in each image of a stereo pair, but the semantic information conveyed by a single pixel is too low to resolve ambiguous matches, therefore we have to consider an area or neighborhood around each pixel. By applying correlation-based matching algorithms to determine the corresponding match, we use local context to resolve ambiguities. The justification for such an approach is that of "continuity," that is disparity values change smoothly, except at a few depth discontinuities. All systems based on area-correlation suffer from the same limitations:

- They require the presence of a detectable texture within each correlation window, therefore they tend to fail in featureless or repetitive texture environments.
- They tend to be confused by the presence of a surface discontinuity in a correlation

window.
- They are sensitive to absolute intensity, contrast and illumination.
- They get confused in rapidly changing depth fields (vegetation).

For these reasons, the existing systems, specially the ones used in "automatic" cartography, require the intervention of human operators to guide them and correct them. Such systems are described in [7, 4, 8, 9, 2].

5.2.2 Feature-based Systems

The depth information in stereo analysis is conveyed by the differences between the two images of a stereo pair due to the different viewpoints, the differences being most prominent at the discontinuities, or edges. Obviously, matching of features will not provide a full depth map, and must be followed by an interpolating scheme. The common characteristics of feature-based matching techniques are:

- They are faster than area-based methods, because there are many fewer points to consider.
- The obtained match is more accurate, edges can even be located with sub-pixel precision [10].
- They are less sensitive to photometric variations, since they represent geometric properties of a scene.

Henderson [5] considered scenes representing cultural sites (man-made structures) and matched edge points on epipolar lines in the two views. He reduced ambiguity by assuming continuity between consecutive epipolar lines. Marr and Poggio have relied on two apparently simple constraints [11]:

1. Uniqueness.
   Each point in an image may be assigned at most one disparity value. One may note that this assumption is not correct for transparent objects.
2. Continuity.
   Matter is cohesive, therefore values change smoothly, except at a few depth discontinuities.

They first proposed a cooperative algorithm [12] that works very well on random-dot stereograms, but they rejected it to propose one of more heuristic nature, implemented by Grimson [13, 14], that generates good results, given the very few assumptions. Arnold [15] matches edges using local context, and his system seems to perform well on cultural scenes. Finally, Baker and Binford [16] match edges on epipolar lines by using the no-reversal constraint that the order of the match has to be preserved, in addition to uniqueness and continuity. They also consider continuity by examining adjacent epipolar lines. This system appears to perform reasonably on a wide variety of images.

In most of the systems presented above, a considerable saving in search time is obtained by a coarse to fine matching, that is the matching is originally done on a low-resolution version of the image and the results are propagated to the higher resolution version. However, it should be noted that in current implementations, good matches as well as errors tend to propagate from one level to the next.

5.3 THE MINIMAL DIFFERENTIAL DISPARITY ALGORITHM

From the survey conducted above, it appears that feature-based techniques are more appropriate to solve the correspondence problem, but edges as a primitive seem to be too low-level, and a connectivity check is needed to remove spurious matches. High level primitives such as physical object boundaries or surface descriptions would be preferred, however, stereo processing may need to precede the computation of such descriptions. As a step towards higher level primitives, we are using segments. In order to generate them, we fit straight lines through adjacent edge points with a given tolerance of one pixel, using an iterative end point fitting technique. These segments can be described by

- coordinates of the end points
- orientation
- strength (average contrast)

By using these primitives, we implicitly assume the connectivity constraint. When matching segments, we need to allow one segment to possibly match with more than one segment in the other image (i.e. to allow for fragmented segments), even if we wish to preserve unique matches for the individual edge points. Also, instead of considering one epipolar line at a time, we have to consider all epipolar lines in which a given segment appears.

5.3.1 Assumptions and Definitions

We consider a simple camera geometry in which the epipolar plane, defined as the plane passing through an object point and the two camera foci, intersects the two image planes, so defining epipolar lines parallel to the y axis. Therefore, corresponding points must lie on corresponding epipolar lines, that is have the same row value, this is illustrated in Figure 2. We also give a bound on the disparity range allowable for any given segment, let us call it maxd.
Let A = $\{a_i\}$ be the set of segments in the left image and
let B = $\{b_j\}$ be the set of segments in the right image.
Then, for each segment a_i (resp. b_j) in the left (resp. right) image, we can define a window w(i) (resp. w(j)) in which corresponding segments from the right (resp. left) image must lie. The shape of this window is a parallelogram, one median being a_i (resp. b_j), the other a horizontal vector of length 2.maxd, as shown in Figure 3. One can see that a_i in w(j) implies b_j in w(i).
We define the boolean function p(i,j) relating two segments as:

p(i,j) is true if
- b_j overlaps w(i)
- a_i , b_j have "similar" contrast
- a_i , b_j have "similar" orientation

The required similarity in orientation is loose and is a function of the segment length. We have set it to be 25° for long segments and up to 90° for very short segments.
Two segments are defined to have similar contrast if the absolute value of the difference of the individual contrasts is less than 20% of the larger one.
To each pair (i,j) such that p(i,j) is true we associate an average disparity d_{ij} which is the

average of the disparity between the two segments a_i and b_j along the length of their overlap.
We define the two functions S_P and $S_{\sim P}$ as:

$S_P(a_i) = \{j | b_j$ in w(i) AND p(i,j) is true$\}$
$S_{\sim P}(a_i) = \{j | b_j$ in w(i) AND p(i,j) is false$\}$

Similarly, we define $S_P(b_j)$ and $S_{\sim P}(b_j)$. We will also need the value $card(a_i)$, which is the number of elements in the set $S_P(a_i) \cup S_{\sim P}(a_i)$.
It is to be noted that all the functions described above are static, meaning that they are computed only once.

5.3.2 Description

Each possible match is evaluated by computing a measure of the distortion this match provokes for its neighbors,i.e. given that (i,j) is a correct match with its associated disparity d_{ij}, how well do the neighbors agree with this proposed disparity? We compute an evaluation of the match (i,j) and compare to the matches (i,k) and (h,j) for k in $S_P(a_i)$ and h in $S_P(b_j)$. If the evaluation is minimum for (i,j), then j is the preferred interpretation for i and i is the preferred interpretation for j. Formally, we compute the following:
At iteration 1

$$v^1(i,j) = \sum_{a_h \text{ in } S_P(b_j) \cup S_{\sim P}(b_j)} \min_{\substack{b_k \text{ in } S_P(a_h) \\ b_k \neq b_j}} |d_{hk} - d_{ij}| \Big/ card(b_j)$$

$$+ \sum_{b_k \text{ in } S_P(a_i) \cup S_{\sim P}(a_i)} \min_{\substack{a_h \text{ in } S_P(b_k) \\ a_h \neq a_i}} |d_{hk} - d_{ij}| \Big/ card(a_i)$$

At the end of each iteration, we define the sets $Q(a_i)$ and $Q(b_j)$ as
j in $Q(a_i)$ and i in $Q(b_j)$ if $\forall k$ in $S_P(a_i)$, $v^t(i,j) \leq v^t(i,k)$ AND $\forall h$ in $S_P(b_j)$, $v^t(i,j) \leq v^t(h,j)$

For any iteration after the first one, in order to evaluate a match (i,j), we only look at the preferred matches for the neighbors of i and j, if they have any. Formally, the computation of $v^t(i,j)$ becomes

$$v^1(i,j) = \sum_{a_h \text{ in } S_P(b_j) \cup S_{\sim P}(b_j)} \min_{\substack{b_k \text{ in } Q(a_h) \\ b_k \neq b_j}} |d_{hk} - d_{ij}| \Big/ \text{card}(b_j)$$

$$+ \sum_{b_k \text{ in } S_P(a_i) \cup S_{\sim P}(a_i)} \min_{\substack{a_h \text{ in } Q(b_k) \\ a_h \neq a_i}} |d_{hk} - d_{ij}| \Big/ \text{card}(a_i)$$

if the sets Q are not empty, otherwise the computation of the function v is done using the formula for iteration 1.

At the last iteration, only those elements that have a preferred match are considered valid, and a disparity map array is filled using these values. It is interesting to note that this process is absolutely symmetric in the two views and therefore will yield identical results (except for the sign of the disparity) if the two views are interchanged. It is helpful to look at a simple example to understand this process.

5.3.3 Example

Let our 2 views be the ones shown in Figure 1. In absence of any extra information, the correct interpretation is that the 3 points have the same disparity, and the result of the matching is (a_i,b_i) for i in {1,2,3}.
In this example,
$S_P(a_i) = S_P(b_i) = \{1,2,3\}$ and $S_{\sim P}(a_i) = S_{\sim P}(b_j) = \emptyset$.
The array d_{ij} is

0	1	2
-1	0	1
-2	-1	0

Therefore we find:

$v^1(1,1) = (|d_{22}-d_{11}|+|d_{33}-d_{11}|)/3 + (|d_{22}-d_{11}|+|d_{33}-d_{11}|)/3 = 0$

compared to

$v^1(1,2) = (|d_{23}-d_{12}|+|d_{33}-d_{12}|)/3 + (|d_{21}-d_{12}|+|d_{23}-d_{12}|)/3 = 1$

and to

$v^1(1,3) = (|d_{22}-d_{13}|+|d_{32}-d_{13}|)/3 + (|d_{12}-d_{13}|+|d_{11}-d_{13}|)/3 = 2.67$

The calculations are similar for the other pairs, so, at the end of the first iteration, the preferred interpretations are only the correct ones, and further iterations will not alter the results.

5.3.4 Discussion

The criterion used here, namely the minimal differential disparity, has similarities with the edge interval constraints given in [17] and subsequently used by Baker [16], but looser in the sense that it does not require ordering of the edges. Since our criterion does not take ordering into account, a dynamic programming implementation is not possible. Our evaluation function is more informed than Baker's in the sense that it considers all edges in a neighborhood instead of just the predecessor and successor of a given edge.

In order to estimate the complexity of our algorithm, we make the following simplifying approximations:

- The image is square, with r rows and columns.
- The density of segments, d, is constant over the whole image. It is equal to the total number of segments, n, divided by the area of the picture, that is, $d = n/r^2$.
- The distribution is isotropic.
- All segments have length l.

Then, each search window has an area $w \approx 2.l.dmax$, and therefore the number of elements in $S_P \cup S_{\sim P}$ is $s = w.d$.
Since we allow an angle tolerance of 30° for long segments and 90° for short segments, the average number of possible matches for each segments is s/3.
For each a_i, we have to look at s/3 elements in $S_P(a_i)$, then s elements in $S_P \cup S_{\sim P}(b_j)$, then s/3 elements in $S_P(a_h)$, leading to a total number of operations of $N = 2.n.s^3/9$.
This formula can be rewritten as $N = 16/9\ N.d^3\ (dmax.l)^3$, which means that if we are working on different windows of a larger image, the complexity only depends on the number of segments in each such window.
If, however, we are working on different resolutions of a given image, then the value of dmax changes with the resolution. Taking a typical value dmax = r/10, the formula becomes $N = 16/9\ N^4\ (l/10r)^3$.
It is interesting to note that we have a function of degree 4 in the number of segments because we do not impose the order preserving constraint used by Baker; his algorithm is of degree 3 is the number of edge points in each line.

The performance of this algorithm on a few examples is presented next.

5.4 RESULTS

It is difficult to display results of stereo matching meaningfully, especially in a two dimensional picture, since we only generate a sparse disparity map. We will simply show the line segments in the two views that are found to match. We have not been able to master the art of cross-eyed stereo fusion, but since a number of people in the field are good at it, we will present all pairs of images according to its convention, that is the left view is shown on the right and the right view on the left. All results will also be shown this way, without explicitly marking each point and its correspondence. We first started our experiments with very simple line drawings, slightly more complex than the one shown in Figure 1 and the results matched the expectations. In order to remove the ef-

fects of the segmentation procedure on the performance of our matching technique, we hand-segmented the images shown in Figure 4 by tracing the boundaries of the objects on a digitizing table. This image, from Control Data Corporation, is synthetic and has been used by Baker [16] for his experiments. The resulting segments are shown on Figure 5 and Figure 6 displays the results after matching. All the lines that have been matched have the correct correspondence, but some matches are missed. This is due to the fact that when the matcher gets confused by closely competing assignments, it chooses not to assign a label. Also, some edges are not matches because of mistakes in the tracing procedure: we traced the boundaries of some objects in opposite directions in the two views.
For all other examples, edge detection was performed automatically using a technique developed by Nevatia and Babu [18] that finds edge magnitude and direction by convolving the image with edge masks in different orientations (we used 5x5 masks in 6 directions here). These edges are then linked to form boundary curves which are approximated by piecewise linear segments.

Next, consider the industrial part shown in Figure 7, the original resolution is 256 by 256 and the grey levels are encoded in 8 bits. We applied the matching algorithm to two different resolutions of the image, running it through three iterations. We found that no assignment changed after three iterations in our experiments. Figure 8 shows the original edges and Figure 9 displays the results in the above mentioned form. Similarly, Figure 10 shows the segments at half resolution and Figure 11 the results. Looking at the segments one by one, we did not notice any incorrect assignment at either resolution, meaning that we captured the shape of the object, even though the density of edges is much larger than in the previous example.

Another, more complex image is shown in Figure 12. In this image, we have a wide range of disparities, a change of sign in the disparities across the picture, various occlusions, the presence of a repetitive structure (a Rubik's cube) and contrast reversal. We do not expect to get good results with this contrast reversal since one of our preliminary conditions is similarity in contrast, but the other peculiarities are very interesting. We worked at low resolution on the segments shown in Figure 13 to obtain the results shown in Figure 14. The interesting points are the following:

- The elongated vertical blocks in the rear of the image are correctly put into correspondence.
- All the squares of the cube that should be identified are correctly matched. The correct labeling appeared at iteration 2 (at iteration 1, most of them are ambiguously matched.)

The segments at high resolution are shown in Figure 15 and the matching results in Figure 16. We did not use the results at low resolution to guide the matching at high resolution, therefore the elongated block in the rear right is not matched any longer. It is interesting to note that the edges coming from the texture of the wood blocks do not create confusion, but help the matching, on the front cylinder for example. Once again, most assigned matches are correct.

5.5 CONCLUSIONS

This research is far from being in a final state. The initial encouraging results presented here must therefore only be viewed as an indication that the hypothesis of minimal differential disparity may be useful. The critical points that must be examined are:

- Relax the contrast constraint. This may be done by considering not the contrast of an edge, but the intensity values on each side. Edges could then be matched if either their left side or their right side correspond. One may eventually consider an edge as a doublet [14] and match each side separately.
- To refine the formulation of the evaluation formula. Statistical analysis may yield better functions, maybe by introducing a static probability measure to evaluate each match based on similarity of intrinsic properties (length, color, orientation). Also of concern is a more accurate definition of a no-match label, which is obtained if a match pair is not clearly better than the competing ones.
- Further extensive testing is also required on aerial and close range imagery, with terrain models for accuracy checking.
- Finally, we must use an interpolation scheme, very likely intensity-based, to generate a full disparity map of the scene depth.

REFERENCES

[1] D. Gennery, "Object Detection and Measurement Using Stereo Vision," in Proceedings of Image Understanding Workshop, College Park, Md., Apr. 1980, pp. 161-167.

[2] H. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover," Stanford Artificial Intelligence Laboratory, AIM 340, Ph.D. Thesis, Sept. 1980.

[3] R. Kelly, P. McConnell and S. Mildenberger, "The Gestalt Photomapping System," Journal of Photogrammetric Engineering and Remote Sensing, Vol. 43, No. 1407, 1977.

[4] D. Panton, "A Flexible Approach to Digital Stereo Mapping," Journal of Photogrammetric Engineering and Remote Sensing, Vol. 44, No. 12, Dec. 1978, pp. 1499-1512.

[5] R. Henderson, R. Miller and C. Grosch, "Automatic Stereo Reconstruction of Man-Made Targets," SPIE, Vol. 186, No. 6, Digital Processing of Aerial Images, 1979, pp. 240-248.

[6] S. Barnard and M. Fishler "Computational Stereo," ACM Computing Surveys, Vol. 14, No. 4, Dec. 1982, pp. 553-572.

[7] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in Proceedings of Image Understanding Workshop, Washington, D.C., Apr. 1981, pp. 121-130.

[8] Hannah M. "Bootstrap Stereo," in Proceedings of Image Understanding Workshop, College Park, Md., Apr. 1980, pp. 201-208.

[9] S. Barnard and W. Thompson, "Disparity Analysis of Images," IEEE Trans. Pattern Anal. Machine Intell., PAMI-2,4 July 1980, pp. 333-340.

[10] P. MacVicar-Whelan and T. Binford, "Line Finding with Subpixel Precision," in Proceedings of Image Understanding Workshop, Washington, D.C., Apr. 1981, pp. 26-31.

[11] D. Marr and T. Poggio, "A Theory of Human Stereo Vision," Memo. 451, Artificial Intelligence Laboratory, MIT, Cambridge, Mass., Nov. 1977.

[12] D. Marr and T. Poggio, "Cooperative Computation of Stereo Disparity," Science 194, 1976, pp. 283-287.

[13] W. Grimson and D. Marr, "A Computer Implementation of a Theory of Human Stereo Vision," in Proceedings of Image Understanding Workshop, Palo Alto, Calif., Apr. 1979, pp. 41-47.

[14] W. Grimson, "From Images to Surfaces," MIT Press, Cambridge, Mass., 1981.

[15] D. Arnold, "Local Context in Matching Edges for Stereo Vision," in Proceedings of Image Understanding Workshop, Cambridge, Mass, May 1978, pp. 65-72.

[16] H. Baker, "Depth from Edge and Intensity Based Stereo," Stanford Artificial Intelligence Laboratory, AIM 347, Stanford, Calif., Sept. 82.

[17] D. Arnold and T. Binford, "Geometric constraints in Stereo Vision," Society Photo-Optical Instr. Engineers, Vol. 238, Image Processing for Missile Guidance, 1980, pp. 281-292.

[18] R. Nevatia and K. Babu, "Linear Feature Extraction and Description," Computer Graphics and Image Processing, Vol. 13, pp. 257-269, July 1980.

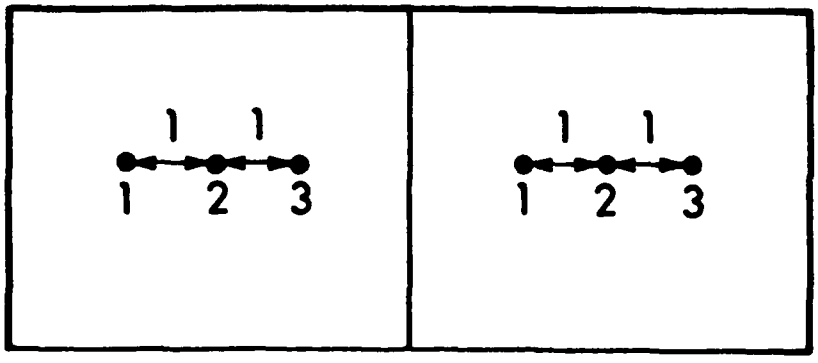


Figure 1: A simple example

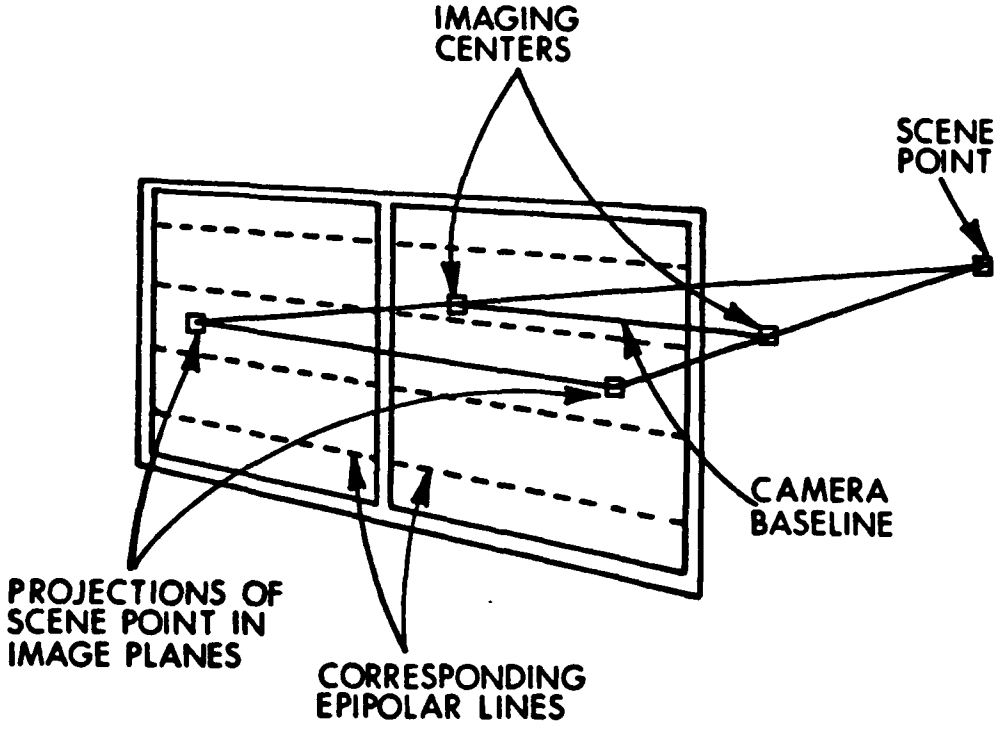


Figure 2: Collinear epipolar geometry from [14]

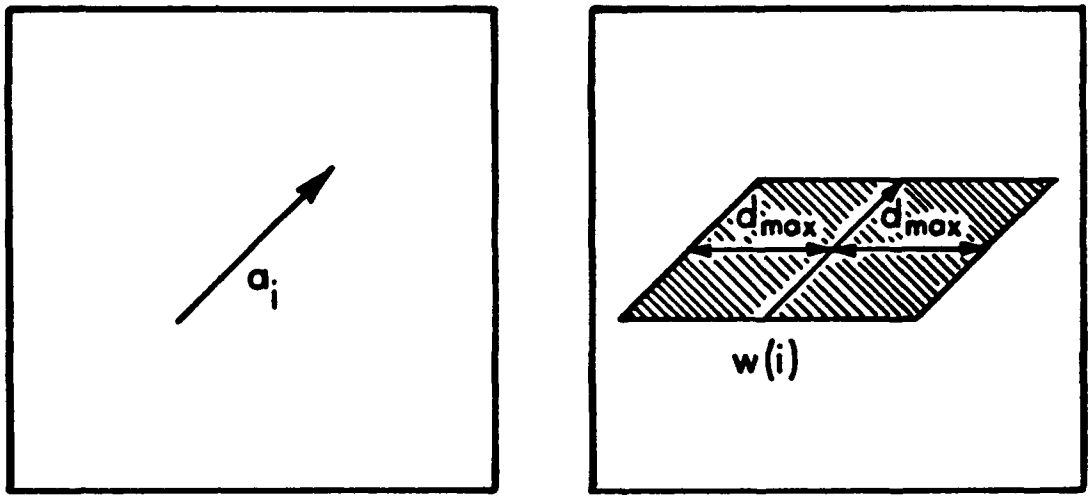


Figure 3: Example of window construction

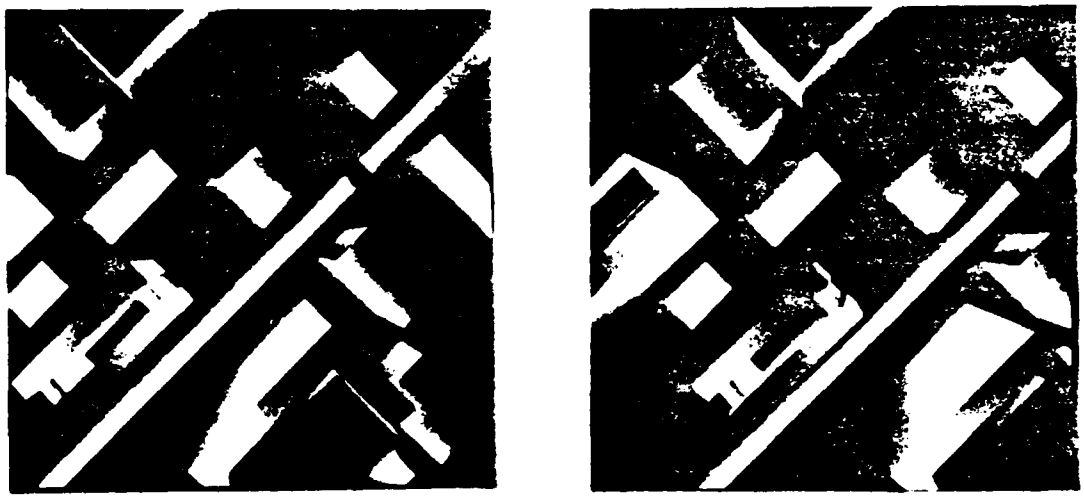


Figure 4: Synthetic image [256x256x6]

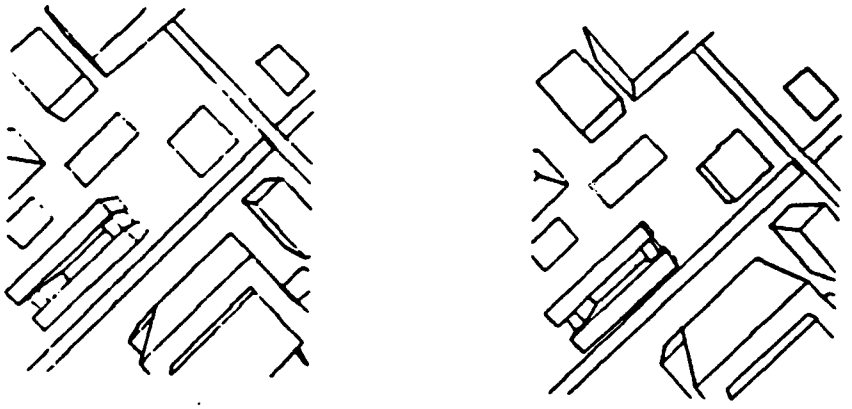


Figure 5: Hand generated segments

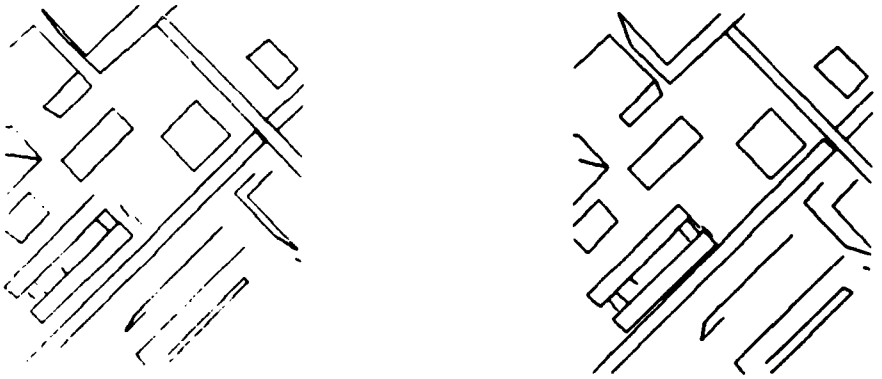


Figure 6: Results of the matching

Figure 7: Industrial part [256x256x8]

Figure 8: Segments from the full resolution image

Figure 9: Results at full resolution

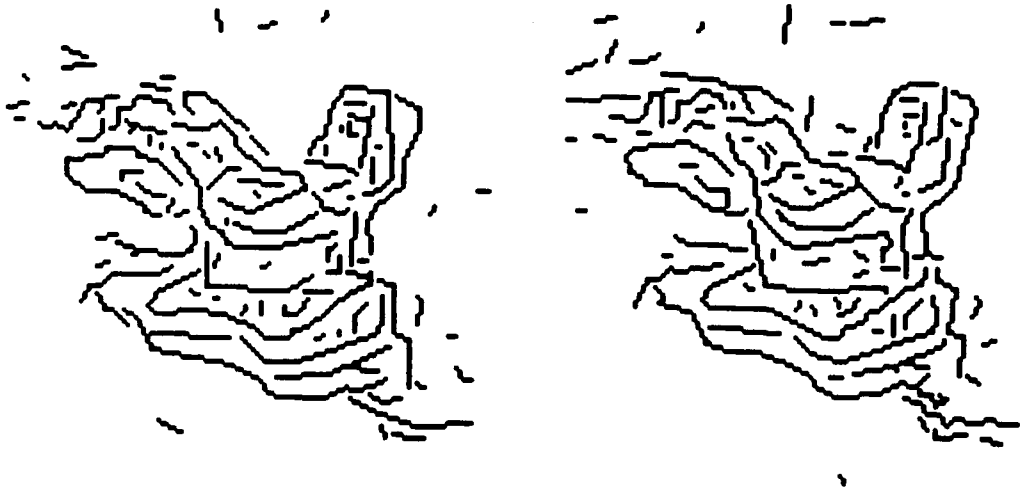


Figure 10: Segments from the half resolution image

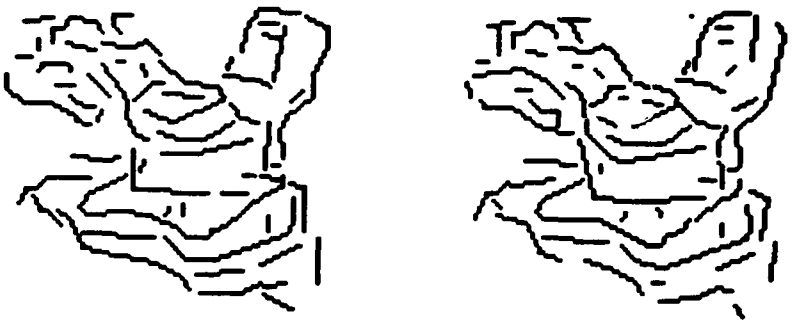


Figure 11: Results at half resolution

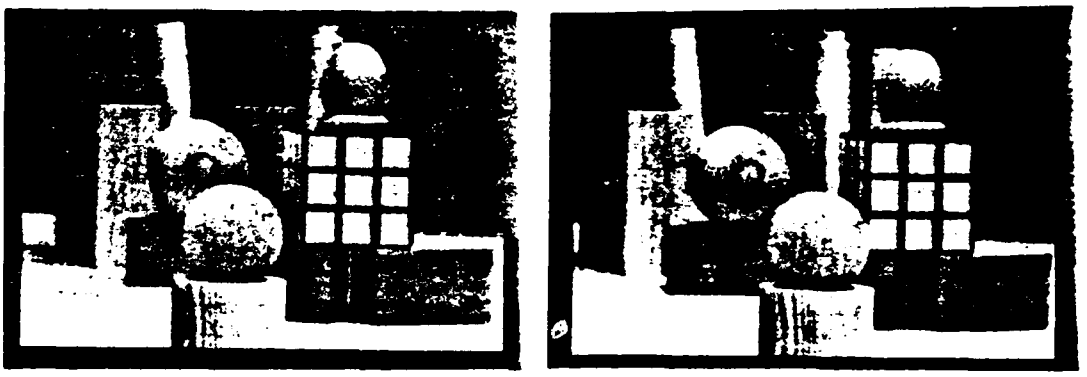


Figure 12: Image of some blocks[512x512x7]

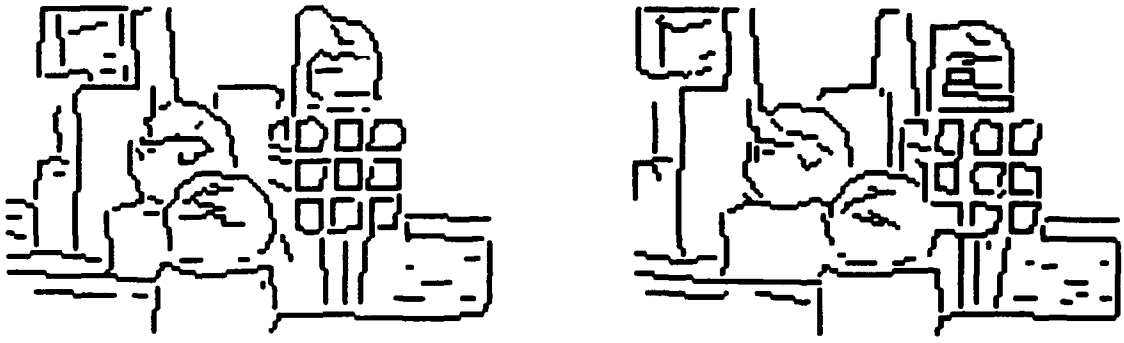


Figure 13: Segments at low resolution

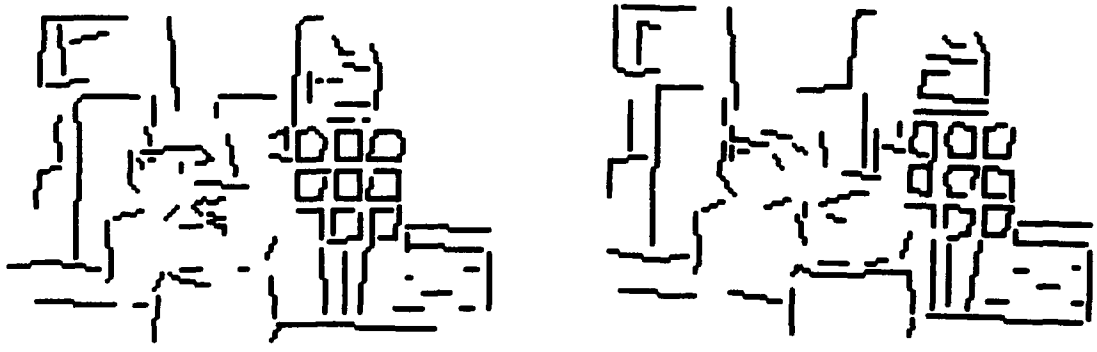


Figure 14: Results at low resolution

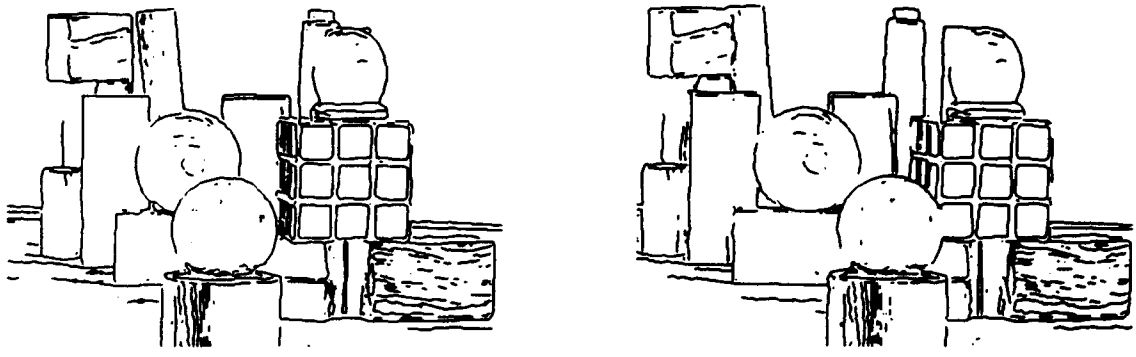


Figure 15: Segments at high resolution

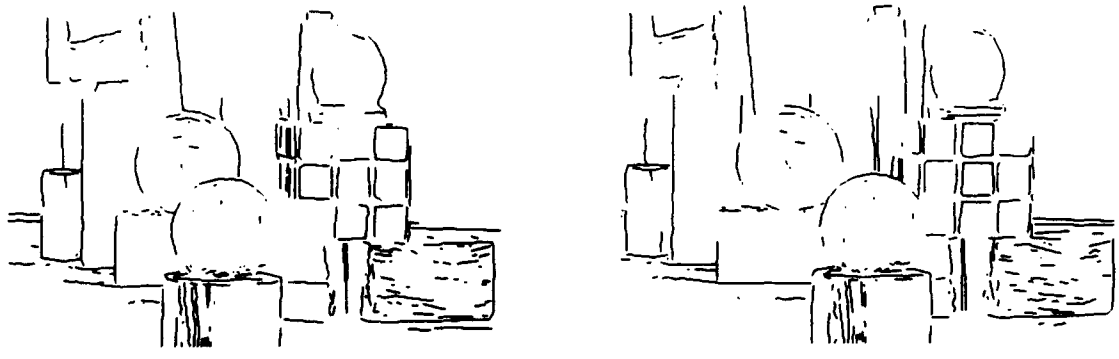


Figure 16: Results at high resolution

SECTION 6

USING SHADOWS IN THE INTERPRETATION OF AERIAL IMAGES

Andres Huertas

6.1 INTRODUCTION

When presented with a single monochromatic aerial image, a human observer perceives a number of clues and features that give a strong impression of depth. In part this is due to the ability to recognize familiar shapes and textures together with other available cues that reinforce the interpretations made. One such clue is due to the shadows cast by the three dimensional objects. Our goal is to show that under simplifying assumptions, it is possible to derive 3-D information from distinguishable objects and shadows, and that 3-D shape hypotheses can be made from distinguishable shadows cast by objects or portions of objects that are not visible or difficult to distinguish.

It is unclear how humans use shadows when interpreting visual information. If we refer to shadow analysis as the process of locating the shadows in the image, establishing the correspondence between shadow casting elements and shadow boundaries, and the use of these pairs to obtain three dimensional information and infer geometric interpretations, then the analysis of the shadows in arbitrary natural scenes requires substantial conscious effort, and therefore can be considered a complex and difficult task. Shadow analysis however, becomes useful when other techniques are not applicable, and simplifying assumptions are made about viewing angles, and either the surfaces casting the shadows or the shadowed surfaces. This is the case with high altitude aerial images where the "third dimension", or objects height, is almost completely lost in the image due to the overhead vantage point. In fact the observed shadows cast often become the main source of three dimensional information from a single aerial image.

In this section we discuss two techniques to obtain information from the shadows which allow us to locate three dimensional objects, and obtain 3-D descriptions of at least the object portions that cast shadows. We assume that the sun is the source of illumination, considered a point source at infinity, and that the sun angles are known a-priori. The ground surface in the immediate surround of the objects of interest is flat and horizontal (or, if there are differences in the elevation of portions of the ground surface, these are small compared to the altitude of the camera). The camera principal ray points to the center of the scene, and the camera is located at a known altitude. We do not assume that the objects have a specific shape, but that that the objects shape correspond to generalized cones with straight vertical axes. The visible top surface of the objects is either flat or consists of flat portions. Also objects do not occlude each other although an object may occlude another objects shadow. Previously we have shown that if the shape of the objects is known in advance, we are able use the shadows cast to detect buildings in aerial images using as a clue the restricted shapes that many of them have. The shadows also allowed us to differentiate the buildings from other objects having similar shapes, and for height estimation [1].

Our techniques use the intensity data and line segment approximations to the intensity edges in the image. They differ in the way the image is initially segmented, which in turn determines the amount of line segments available for analysis and 3-D interpretation. The first technique involves the intensity data and the processing of line segments throughout the image. The second technique involves only the line segments approximating the boundaries of shadow regions extracted from the image. Both techniques automatically assign an initial interpretation to the line segments on the basis of assumptions regarding object geometry, photometry, constraints imposed by the direction of illumination and the position of the camera. Line segments are grouped into classes according to their interpretation labels. From these classes we form graph structures that represent the correspondence between shadow and object lines, and boundary continuity along object and shadow outlines.

From the established correspondences and other evidence of shadow boundaries cast by visible or non-visible object portions, we are able to make inferences about the object portions casting shadows as well as the shape of the visible or non-visible object side surfaces. The resulting inferences are shown by means of a 3-D wire model of the objects in the scene for the first technique, and a 3-D wire frame model of the shadow casting object portions for the second technique.

Previous work on the interpretation of image edges [2,3,4] and processing of scenes with shadows [2] is based on boundary junction constraints which are difficult to obtain from natural images. Theoretical work on shadows by Shafer and Kanade [5] has concentrated in obtaining surface orientation, and no experimental results have been reported. Binford and Lowe [6] discuss the derivation, use and implementation of more general constraints to carry out geometric interpretations up to the volumetric level. Their technique has been tested on simulated image edge data derived by hand; it requires accurate curve junction information, which may be difficult to obtain from natural images. With a similar philosophy, we use edge and region data derived automatically which explicitly implies dealing with partial information. Hence, further assumptions about the objects are introduced that may limit generality. We make extensive use of the inherent redundancy which results from the fact that adjacent shadow casting object points cast adjacent shadow points, to predict the presence of weak evidence as well as to hypothesize missing data resulting from errors in the data, poor contrast, inadequate sensor response, and and other scene and illumination conditions that prevent automatic boundary detection.

We consider the analysis of the shadows cast not a complete solution in itself, but a means to obtain information that can be used in conjunction with other 3-D surface inferencing techniques, such as intensity and feature-based stereo, and local shape from shading. This is particularly important if we are interested in detecting cultural features such as buildings and other 3-D man-made objects.

6.2 PRIMITIVES FOR SHADOW ANALYSIS AND SHADOW GEOMETRY

Our shadow analysis techniques are segment-based, segments being line approximations to the intensity boundaries in the image. Image boundaries seldom have a unique interpretation locally (true physical occluding or non-occluding convex and concave edges, apparent edges, reflectance edges, illumination edges) but their stability with respect to geometry, illumination and sensor is quite general and useful. We define a shadow model, analyze the image, and obtain three dimensional descriptions in terms of these primitives.

We define shadows as portions of 3-D space not illuminated by direct sun light due to an interruption in the flow of illumination radiation by an occluding object in 3-D space. As a result the occluding contour is projected onto the surface(s) beneath the object (from the source vantage point) forming dark regions surrounded by illumination discontinuities. If we think of this space as a shadow "solid", then it consists of two terminators joined by an imaginary illumination surface generated by the light rays passing through corresponding shadow casting points along the occluding contour, and shadow cast points along the shadow boundary. A similar definition is given in [5]. We call an S-terminator the shadow cast, and an O-terminator the self-shadowed surfaces of the object casting the shadow. Figure 1 shows a planar and a curved surface (O-terminators) and the shadows (S-Terminators) cast by these surfaces, illuminated from above and behind by an infinitely distant light source.

Under the flat ground assumption, S-terminators are planar, parallel to the image plane and hence, visible from the camera vantage point. O-terminators correspond to object sides and, under perspective, may be visible, depending on the position of the camera. As a result, since the sun angles are known, only the O-terminator parameters are left to be determined or hypothesized for each shadow solid.

Given a visible shadow casting boundary along the top surface of an object and its corresponding shadow boundary, those visible points at the extreme ends of the shadow casting boundary are denoted object extreme points (OEP). Corresponding points, in the direction of illumination, on the matching shadow boundary are denoted shadow extreme points (SEP).

With the stated definitions and assumptions we are able to model the imaging process and shadow geometry in a straightforward manner. Basically, the sun rays are parallel and strike the objects at a known angle and with a known direction. Points along some of the objects boundaries, constrained by the sun angles, will cast corresponding shadow points. These shadow points will be located in the image plane in a certain direction (see below) from the shadow casting points, according to the perspective transformation effected by the camera lens. Figure 2 shows the geometry.

Let (X,Y,Z) be the scene reference coordinate system and, (Xc,Yc,Zc) the camera coordinate system. With the camera at (0,0,Ac) and its principal ray pointing down along the Zc axis, the vertical vanishing point will be located at (0,0,-Ac). (Therefore, all visible vertical edges in the image must be oriented towards the origin of the camera coordinate system).

Let α be the direction of the illumination rays projected on the ground surface and, i, the sun incidence angle. If we associate a coordinate system (Xs,Ys,Zs) with the Xs axis along the ground projection of the sun ray passing through the origin of the camera coordinate system, then we determine the direction of the projection of the 3-D sun rays onto the image plane, by computing for a given (x,y) point the unit vector I, given by

$$I = \left[\frac{I_x}{|I|} , \frac{I_y}{|I|} \right] , \text{ where}$$

$$[I_x, I_y] = [Ac - x_s Cot(i) , -y_s Cot(i)], \text{ and}$$

$$|I| = \sqrt{I_x^2 + I_y^2}$$

This simple expression requires that the Xc axis be aligned with the Xs axis, and that the image point (x,y) be expressed in (Xc,Yc) coordinates. The later transformation to obtain (x_c, y_c) from (x,y) is a translation, and axis alignment gives (x_s, y_s) by rotation:

$$x_s = x_c Cos\alpha + y_c Sin\alpha$$

$$y_s = -x_c Sin\alpha + y_c Cos\alpha$$

To obtain the vector I in image coordinates, we rotate its components by $-\alpha$. Hence, *for any given image point, I gives the imaged direction of the sun ray passing through* that point. Establishing object-to-shadow segment correspondences under perspective is along these I vectors, as discussed in the following sections.

6.3 THE SHADOW ANALYSIS PROBLEM

6.3.1 Locating the Shadows in the Image

(a) Discussion

The problem in locating cast shadows is to distinguish the transitions between direct and scattered illumination from changes in reflectance or surface orientation. Since these parameters contribute together to the image, this discrimination is difficult; it requires that the effects of each parameter be distinguished from one another.

One technique to distinguish shadow and occluding edges from other edges [7] suggests that the boundaries of shadow S-terminators are signified by a high correlation across the edge, with an abrupt shift in the regression parameters in the neighborhood of the edge. With this approach, high resolution is required to make accurate measurements and the technique has been tested on hand derived data only. This technique may be useful for fine and detailed object description and recognition. We make use of a weaker constraint: If there are visible shadows in the image, in the sense that they are distin-

guishable from the background, then the visible S-terminators are darker than the background. Reflections from nearby surfaces complicate matters but, with the sun as source of illumination, the ratio between shadow and sunlight will be the ratio of diffuse to direct sunlight, which is roughly equal across shadow boundaries within an image [8].

To locate the S-terminators or their boundaries (see both methods below), we first determine from the intensity data the range of gray levels that a shadow region in the image is likely to have, on the basis of the assumption that most of the area in the image corresponds to smooth and continuous regions. We have previously shown [9] that by masking out the portions in the image of a sunny scene where the intensity transitions occur (edges), it is possible to obtain a more accurate histogram of the distribution of the gray levels of the regions in the image. Most shadow gray levels are in the "dark" side of the masked histogram and, depending on the illumination conditions, histogram features can be located to provide a reliable shadow gray level range, as shown in figure 3.

(b) Locating the shadows in the edge-based method

In our edge-based method we first compute line segment approximations to the intensity boundaries in the image using the Nevatia-Babu technique [10], and obtain a list of records of line segment attributes, such as segment ID, the coordinates of their "begin" and "end" points, angle of orientation, relative contrast and its predecessor and successor line segments. The direction associated with a line segment indicates that the brighter side is to the right. Next we automatically assign an initial interpretation to the line segments by classifying them into one or more of the following classes:

- Type 1: Shadow casting line
- Type 2: Non-shadow casting line
- Type 3: Object line not on top surface
- Type 4: Shadow line cast by a non-vertical line
- Type 5: Shadow line cast by a vertical line
- Type 6: Shadow occluding line
- Type 7: Vertical line
- Type 8: Reflectance line

Figure 4 show the intended classification for an object viewed from the same camera position but illuminated from two different light source positions; the I and C arrows represent the projected illumination direction and a camera ray, respectively. The segments are classified on the basis of photometry (edge contrast and gray level statistics in the neighborhood of the segments), assumptions regarding object geometry and constraints imposed by the direction of illumination and camera position. The following are the heuristic rules used in our current implementation:

- Type 1: If the light source is on the brighter side of a high contrast line segment and, the darker side is likely a shadow (i.e. the average gray level of the region adjacent to the segment is within the shadow gray level range), then the line segment is classified as a shadow casting segment. If the contrast is low, these lines are also classified as shadow casting although depending on the position of the camera,

the O-terminator may be visible and these lines may correspond to object lines not in the boundary of the object top surface.

- Type 2: If there is no evidence of a shadow S-terminator on the darker side of the segment and one of the regions adjacent to the segment is smooth (has a small intensity variance), then the segment is classified as a non-shadow casting line.
- Type 3: If the light source is on the brighter side of a low contrast line segment and, the darker side is likely a shadow then the segment is classified as an object line not on the object top surface shadow casting boundary. In fact, these lines may in the boundary between an O-terminator and an S-terminator. If the region on the darker side is not a shadow but one of the regions adjacent to the segment is smooth and bright then depending on the combined illumination direction camera position, this segments are also classified as lines not in the object top surface.
- Type 4: If the source of illumination is on the darker side of the segment and, the region on the darker side is likely a shadow S-terminator then the line segment is classified as a shadow segment.
- Type 5: If the segment is parallel to the projection of the sun rays on the ground, and the region adjacent to the darker side is dark enough to be a shadow S-terminator, then the segment is classified as a shadow line cast by a vertical line in the boundary of the O-terminator. The vertical boundary is visible or not depending on the position of the camera. This constraint also allows us to determine the general position of the object casting the shadow.
- Type 6: If the source of illumination is on the darker side of the segment, the region on the darker side is likely a shadow S-terminator, and the region adjacent to the brighter side of the segment is smooth then the line segment is classified as shadow occluding line.
- Type 7: If the segment is oriented towards the vertical vanishing point, which coincides in the image with the origin of the camera coordinate system, the segment is classified as a vertical line.
- Type 8: If there is no evidence of a shadow S-terminator on the darker side of the segment and the segment can not be classified into another class then it is classified as a reflectance line.

With our classification scheme there is seldom a unique interpretation for each line segment but in some cases more than one interpretation is valid, due to coincidences resulting from overhead viewing. A visible shadow boundary may coincide in the image with an occlusion boundary line obscuring the shadow cast by a nearby object. Misinterpretations and conflicts are resolved when we look for object-to-shadow correspondences and attempt to derive global interpretations.

(c) Locating the shadows in the region-based method

In our region-based method we use the range of shadow gray levels to extract the dark regions in the image, using a recursive splitting method [11]. Next we approximate the region boundaries with line segments. The segments obtained are then automatically classified into a subset of the classes in the above method, namely, types 1, 4 and 5; the other classes do not occur since we have only the boundaries of the shadow regions.

6.3.2 Establishing Object-To-Shadow Correspondence

(a) Discussion

Medioni [12] has reported an approach to the correspondence problem which assumes knowledge of the position of a shadow casting object. Given the outline of the object, he looks for a the outline of a dark region adjacent to the object in a direction consistent with a known direction of illumination. Correspondence is established by finding a vector that best describes the correspondence between object and shadow. Such vector is found by using the maximum peak of the correlation function of the region and its shadow.

Our techniques do not assume knowledge of the position of the object but rely on the initial interpretations assigned to the segments to locate object and shadow outlines, and uses the illumination constraints to establish explicit object-to-shadow correspondences along illumination vectors, with the following assumptions:

- Object surface boundaries are continuous and the visible shadow boundaries cast by object boundaries are also continuous.
- Straight boundaries in the image correspond to continuous straight boundaries in the scene. This fails if a curved physical boundary coincidentally lies in a plane aligned with the camera. This is detected if the boundary casts a curved shadow boundary and the shadow is not occluded by a curved object.
- A shadow line has a corresponding shadow casting line, visible or not, and this shadow casting line lies between the shadow line and the source of illumination in 3-D space.
- Parallel lines in the image correspond to parallel lines in the scene. This is not true if there is an accidental alignment of the camera with the planes that contain them.

(b) Establishing Correspondences in the Edge-based Method

So far we have eight classes of line segments. Our immediate problem is to bring into correspondence the shadow casting segments and the shadow segments and create a record of these correspondence. Since we expect fragmentation along automatically detected boundary contours, it is difficult to simply collect chains of shadow casting segments and chains of shadow boundaries and try to match entire contours. But we can expect to be able to correspond single segments to a small group of segments. This suggests constructing subgraphs that are combined with other subgraphs (on the basis of boundary continuity), to form a graph that represents the entire matching object and shadow contours.

In our current implementation we have simplified the above idea by performing the search from object-to-shadow segments rather than from shadow-to-object or in both ways. The only reason for this choice was that in our experiments, the class of shadow casting segment was significantly smaller than the class of shadow segments. As a result, the structures for representing the correspondence becomes simpler, i.e. trees

rather than subgraphs. Nevertheless, the extension from trees to graphs should be straightforward. We now discuss the process in some detail. Consider the object and shadow segment boundaries depicted in figure 5a:

i) Construct Correspondence Trees (C-trees, figure 5b). For the length of each shadow casting segment (type 1) we search along illumination vectors for a shadow segment (type 4). The root of the C-tree is the shadow casting segments, and has one leaf node for each distinct shadow segment found. The distances are recorded, and the arcs of the tree represent object-to-shadow segment correspondence.

ii) Combine C-trees into Correspondence graphs (C-graphs, figure 5c). C-trees are combined in an attempt to form a C-graph representing complete matching object and shadow contours. C-trees are combined on the basis of boundary continuity (shown as double lines) as follows: C-trees having leaf nodes with the same shadow segment; C-trees having leaf nodes with adjacent shadow segments; and C-trees having nodes with adjacent shadow casting segments. Notice that this step indirectly implements boundary colinearity and curvilinearity where fragmentation occurs.

iii) Locate SEP and OEP extreme points (figure 5d). The C-graphs so far encode object and shadow continuity horizontally, object-to-shadow correspondence vertically, and represent continuous object and shadow contours. Therefore, the extreme shadow points (ESP) and extreme object points (EOP) are found at the extremes of these contours. If a pair (OEP,SEP) is found to correspond along an illumination ray, then the line segments in the image located between the SEP and the OEP represent the boundary of the shadow cast by the side (possibly non-visible) of an object, whose shape can be derived from the shadow information. We discuss the information that can be derived from these segments in the next subsection. At this point we only determine whether these segments indicate that the object sides are vertical or not. This is rather straightforward since vertical edges cast shadow edges parallel to the projected direction of illumination, and these segments are readily available in the class of segments of type 5 (v_1, v_2 and v_3 in figure 5a). This information allows us to determine the relative position of the object, including the non-shadow casting portion (see below). The evidence is added to the C-graphs on the basis of continuity, as shown in figure 5d.

iv) Locate non-shadow casting segments. We know that the shadow casting and non-shadow casting outlines of the object top surface converge at OEP points. To locate non-shadow casting contours, we form non-casting chains (NC-chains) with segments of type 2. Next, the following rules are applied:

- If the end points of a chain coincide with the OEP points in some C-graph, then the chain represents the non-shadow casting portion of the top surface of an object. Note that depending on the position of the camera, the sides of an object may be visible and more than one chain can be found. The correct interpretation for these chains is discussed in the next subsection.
- If the end points of an NC-chain coincide with OEP points at the extremes of two different C-graphs, then have three cases:

  *An OEP was not located for one of the two C-graphs due to shadow occlu-

sion (see question marked node in figure 5d). In this case, there is a missing NC-chain between the two C-graphs. They are combined into a complete C-graph on the basis of boundary continuity after the missing chain is found (see the O_5-O_6 chain in figures 5e and 5f).

*An OEP was located for each of the C-graphs. Usually one of these is missing due to shadow occlusion, and the other, due to missing segment information. The C-graphs are combined on the basis of boundary continuity evidence as in the previous case. If no evidence is found, then the overall shape of the object outline can be used to hypothesize the missing information. In our current implementation this step is limited to hypothesizing a straight line between the "loose" end points.

*The OEP for both C-graphs were located. In this case, the graphs are also combined on the basis of continuity but there must be another C-graph between the two C-graphs. The second case occurs for an unusual combination of illumination direction and object geometry and position.

- If only one end point in an NC-chain coincides with an OEP in a C-graph for which both OEPs have been located, then we for evidence of continuation between the chain "loose" end point and the remaining OEP. These evidence may be found in the classes of segments of type 2 and type 8. As before, if no evidence is found, the missing portion is hypothesized.
- Further combination of trees as well as the hypotheses for missing or unseen portions of the outlines can be performed on the bases of proximity, colinearity, curvilinearity and symmetry. This step has only been partially implemented.

(c) Establishing Correspondences in the Region-based Method

In the region-based method we only have closed shadow contours, with the segments along the contours classified into three classes: Shadow segment (type 4), shadow segment cast by a vertical edge (type 5), and "other" which we have initially classified as having type 1. The information that we derive from the shadow regions primarily depend on whether the "other" class in fact should be labeled type 1 or whether it should be labeled type 4. In the former case, the region boundaries correspond to the boundaries of an S-terminator whose segments labeled with type 1 correspond to coincident O-terminator boundaries. In the latter case, no O-terminator boundaries are available. If we determine that the case for a given shadow region is the former, then we attempt to establish object-to-shadow segment correspondences between type 1 and type 4 segments. Otherwise we do not attempt to establish a correspondence.

In order to determine whether correspondences are to be established we apply the following two heuristic rules:

- If the amount of "orthogonality", Ω, among the segments in the region contour is high then attempt this region is a good candidate for attempting correspondence.
- If the segments along a region contour consists of chains of segments of type 4 (possibly mixed with segments of type 5) and chains of segments of type 1, rather than a random mix of segments of types 1,4 and 5, then this region is a good can-

didate for attempting correspondence.

If a region is found to be a good candidate according to the first or both rules, then correspondence is attempted. Let us now elaborate in some detail about these rules and how we apply them, with an example, and with the derivation of the Ω measure.

Consider the shape of the shadows cast by a rectangular building such as the one shown previously in figure 5a. Most of the shadow outline consists of straight and long portions, there are many pairwise parallel segments in the boundary as well as several 90° corners. Segments S_1 to S_3, and segments S_4 to S_7 form two continuous chains of type 4 segments. Segments O_1 to O_4 and segments O_7 to O_9 form two chains of type 1 segments. Segments V_1 to V_3 are of type 5. The remaining segments, O_6, O_{10} and O_{11} would not be available in this method. We would have two separate shadow regions and clearly both would satisfy the above two rules.

Consider now the hand drawn shadow cast by a tree depicted in figure 6a. The arbitrary mix of segments with types 1 and 4, as well as the low degree of orthogonality among the segments in the boundary clearly fail the above rules.

Let us now derive the Ω measure and discuss the accuracy of the above rules. To compute Ω for a given shadow region:

1) Compute the function f(θ), equivalent to a length weighted orientation histogram for the segments in the boundary.

$$f(\theta) = \sum_{k} S_L(k) \cdot S_\theta(k) \text{ where,}$$

- k is a line segment in the boundary of the region.
- $S_L(k)$ is the length of segment k.
- $S_\theta(k)$ is the orientation of segment k such that $S_\theta(k)$ MOD 90° = θ.

2) Determine the predominant segment orientation θ_{max}.

$$\theta_{max} = \text{MAX } [f(\theta)] \text{ with } 0 \leq \theta < 90°.$$

3) Compute Ω, the degree of orthogonality as a percent.

$$\Omega = \sum_{\theta_{max}+\varepsilon}^{\theta_{max}-\varepsilon} \frac{100 \cdot f(\theta)}{\sum_{k=1}^{segno} S_L(k)} \text{ where,}$$

- segno is the number of segments in the outline.
- ε is a small angle tolerance to compensate for angle distortions in the line segment

approximations.

For a perfect rectangle or combination of rectangles with boundaries along two orthogonal directions, $f(\theta)$ is zero everywhere except at $\theta = \theta_{max}$; θ_{max} coincides with the orientation of the longer sides modulo 90° and, $f(\theta)$ will have a value equal to 100%. On the other hand, a circular region with very short boundary segments will result in an evenly distributed $f(\theta)$ and thus, a very small Ω. Note that the measure also encodes parallelism since 180° is a multiple of 90°.

The question remains as to what constitutes a random mix of segment labels along a region boundary, and what constitutes a high or low degree of orthogonality, or whether other known shape descriptors would be more adequate. Take the convex hull approach. Suppose that we compute the convex hull for one of the shadow regions of the building shown in figure 5a, and the convex hull for the shadow of the tree shown in figure 6a. If we decide that a region whose shape is close to the shape of its convex hull (as in the tree shadow) is not a good candidate, then the building shadow would be a good candidate. But this would fail for many man-made objects having cylindrical shapes, or for objects having parallel side and illuminated from one side. Other measures would tell if the region shape is elongated or not but still not capture the geometric detail needed. Our experiments with images containing man-made objects such as vehicles, fuel storage tanks and conventional building structures have yielded orthogonality measurements well above 50%, while objects like trees tend to yield measurements well below 50%. As a result our criteria uses 50% to make a decision.

An arbitrary mix of type 1 and type 4 segments indicate a succession of large changes in the orientation of the segments along the region boundary. If the region is in fact a shadow region, the illumination occluding profile also has this changes in orientation along its boundary. This is a good indication of a object having complex surfaces, such as a tree. On the other hand, a mix may indicate that the segments have been assigned incorrect labels but these would tend to be sporadic or occur systematically. We have not studied ways to detect systematic mixes but we have observed that in many cases, a mislabeled segment along the boundary of a region with a high degree of orthogonality and parallelism is brought in to correspondence with a segment that is also mislabeled. In fact, interchanging the labels would result in the correct labeling. As a result we rely more heavily on Ω than on the labeling mix. We currently consider the mix criteria valid only for regions with low Ω.

6.3.3 Geometric Interpretation of Object/Shadow Pairs

(a) Discussion

The most obvious information that can be derived from the established correspondences is the height of the shadow casting elements. In the absence of occlusion by other objects, the observed shadow width in the image can give an accurate estimate of the object height, which is equivalent to the shadow width multiplied by cotan(i), where i is the sun incidence angle. If there is occlusion, a lower bound on the height is obtained.

Our problem now is to derive the shape of the O-terminator surfaces by geometric reasoning, involving the information in the C-graphs, the segment configurations between OEP/SEP pairs, and the segment junctions involving shadow segments. Consider the following assumptions:

- Under the flat ground assumption a constant shadow width along corresponding object and shadow boundaries indicates constant elevation of the shadow casting points. A smoothly varying shadow width indicates a slanted object top surface, or shadow occlusion.
- Breaks in the shadow (L-junctions) boundary correspond to breaks in the shadow casting boundary. Otherwise, abrupt changes in shadow width indicate an O-terminator aligned with the observer or not visible. The amount of change is proportional to the amount of change in elevation of the shadow casting elements.
- On a flat shadowed surface, shadow boundaries lying between an OEP and a SEP in the 2-D image which are parallel to the projected direction of the illumination have been cast by a vertical edge, visible or not.
- Since the ground is flat, a straight shadow casting boundary cannot cast a curved shadow boundary. This fails if the observed shadow lies on, or is occluded by a curved object in the image, or, if a curved shadow boundary has been cast by a curved shadow casting boundary lying on a plane aligned with the camera.
- Parallel lines in the image correspond to parallel lines in the scene. This fails if there is an accidental alignment of the camera with the planes that contain them.
- Given a pair of matching object and shadow contours, the shape of the O-terminator is derived from the segments located between corresponding OEP and SEP pairs at the ends of the matching contours.

(b) Obtaining 3-D Interpretations in the Edge-based Method

In this technique, the segments between corresponding OEP/SEP pairs determine the shape and position of the O-terminators in 3-D space, and the T-, Y-, and ↑ junctions at the end of the shadow contours determine the geometric relationship between the O-terminators and the top visible surfaces of the objects.

Figure 7 shows the basic shape hypotheses that are derived from the T-, Y- and ↑ junctions and the segments between OEP/SEP pairs. The arrowheads denote shadow lines of type 5 (cast by vertical lines). In figure 7a the segments between the two corresponding OEP/SEP pairs signify a vertical O-terminator, and the ↑ junction signify that the shape of the O-terminator is constant from the top of the object to the ground surface. In figure 7b, the segments between the OEPs and the SEPs indicate that the top structure of the object has vertical sides and that this structure is supported by a smaller structure whose sides do not cast visible shadows. The distance between the end of the stem of the T-junction and the OEP determines the elevation of the top structure from the ground. The T-junction indicates occlusion but also that there is no information available to determine the shape of the support structure. Figure 7c shows a similar case but the absence of segments of type 5 indicate that no information can be derived besides the elevation of the top visible surface. Figures 7d, 7e and 7f show that Y-junctions give information about support structures that are smaller than the top visible

surface.

Object sides are visible under perspective. As a result, barring coincidental alignments, the 2-D junctions at the ends of the shadow contours do not appear geometrically symmetric even if they are so in 3-D (see figure 4). The complete analysis of the possible junctions can become a difficult task, even with complete edge data. In our implementation, we concentrate only on the simpler junctions mentioned above, which are found at one of the end points of a shadow contour. At the other end, we only look for segments of type 7, representing vertical lines, to verify previously made O-terminator shape hypotheses.

An additional source of information for verification of O-terminator hypotheses are the visible object lines which are neither vertical nor in the outline of the top surface (segments of type 3). Most of these lines are difficult to locate in advance since their photometric properties depend on the combination of illumination and camera ray angles. If these lines do not occur as additional NC-chains, then we attempt to locate them only for those objects whose sides have been found to be vertical. We search for segments, from the outline segments in the top surface, in the direction of the vertical vanishing point.

(c) Obtaining 3-D Interpretations in the Region-based Method

In this method we have so far a record of the correspondence between the segments in the boundary, with one C-graph for each shadow region. If no correspondence was established (according to Ω) then the C-graph is empty (see below). Our problem is to derive the shape and position of the O-terminators in 3-D space.

The only type of line junctions in the region boundaries are L-junctions. We distinguish three types of L-junctions depending on the angle formed by the interior angles of the region shape: Narrow (less than 90°), square (90°) and wide L-junctions (more than 90°).

The square, narrow and wide L-junctions correspond to the T, Y, and ↑ junctions discussed earlier, without the "leg" that would have corresponded to an object boundary.

Figures 8a-8f depict only the S-terminators shown previously in figures 7a-7f and the O-terminator hypotheses that are derived. As before, the arrowheads denote shadow lines cast by vertical lines. Narrow L-junctions formed by segments of type 5 and type 1 indicates that a vertical O-terminator appears to be in contact with the ground. If there is a chain of segments of type 1 which is parallel to a chain of segments of type 4, and there is no difference in their length, as shown in figure 8a, then the support structure under the top surface has the same size as the top surface at this portion of the object; otherwise the object has a support structure which is smaller then the top surface, as shown in figures 8d and 8e. Also, the difference is equal to the distance between the junction point and the OEP, and the end point of the chain of segments of type 4 is the SEP.

Square L-junctions, as T-junctions before, may signify occlusion. As shown in figures 8b and 8c, the square L-junctions signify that portions of the O-terminator are off the ground surface. They also indicate that there is no information about the structure supporting the top surface. Wide L-junctions occur throughout the shadow boundaries and have meaning only within the context where they occur; between OEP/SEP pairs they indicate the shape of the shadow casting lines in the sides of the objects, and within chains of segments of type 4 along a shadow boundary, they indicate changes in elevation of the hypothesized shadow casting segments in the O-terminators.

If the C-graph is empty for a shadow region, as it probably would be in the case of the shadow cast by a tree, then we hypothesize from the shadow information a planar O-terminator, perpendicular to the ground surface, whose outline approximates the outline of the illumination occluding profile that cast the shadow. We compute the equation of a line passing through the portion of the shadow boundary that is closest to the source of illumination and parallel to the projected direction of illumination, and compute the distances to this line from the elements in the boundary of the shadow to hypothesize their height, as shown in figure 6.

6.4 RESULTS

We have tested our techniques on small portions of high resolution aerial images, and we used our previously reported segmentation techniques to compute line segments [10,11,13]. Figure 9 shows the intermediate and final results obtained by processing an image with the edge-based method. The entire process is automatic, although it consists of several independent modules that run in sequence. Figure 9a shows a 256x256 picture of fuel storage tanks and figure 9b shows the line segments extracted from it. The classification rules are able to classify 62% of the detected line segments into classes 1-7. Of those, 77% are in a single class, 27% are in two classes, 22% are in three classes, and 4% are in four classes. Figures 9c to 9i show the segments in each class for this image, with the segments of types 1 and 3 combined in (c).

Figure 9j shows the extracted object and shadow boundaries, and figure 9k shows a three dimensional wire frame model for the objects in the scene, which together with the following description of the objects is obtained from the C-graphs:

a) Perimeter
b) Edge segments in outline
c) Approximate center
d) Average width of shadow
e) Approximate height
f) Shadow Occlusion

	OBJECT:1	OBJECT:2	OBJECT:3	OBJECT:4	OBJECT:5	OBJECT:6
a)	84.3	40.2	72.2	135.6	105.8	84.8
b)	10	6	11	17	14	11
c)	(72,125)	(56,78)	(93,93)	(111,127)	(137,79)	(155,113)
d)	20.9	15.0	14.5	14.8	12.7	13.5
e)	36.2	25.9	25.1	25.7	22.0	23.5
f)			Occlusion by obj:9	Occlusion by obj:3	Occlusion by obj:1	Occlusion by obj:5

	OBJECT:7	OBJECT:8	OBJECT:9	OBJECT:10	OBJECT:11
a)	99.6	141.2	51.9	42.0	99.5
b)	15	18	8	9	13
c)	(170,141)	(210,100)	(80,62)	(79,226)	(114,47)
d)	8.1	17.5	15.7	16.8	17.4
e)	14.1	30.3	27.2	29.1	30.2
f)	Occlusion by obj:6				

Figures 10 and 11 show the intermediate and final results obtained with the region-based method. Figure 10a is the 128x128 original image showing a building and some trees. Figure 11a is the 300x300 original image showing several vehicles, two trees, and a lamp post casting shadows. Figures 10b and 11b show the line segments in the boundaries of dark regions extracted automatically from the original images. The left portions of the shadows of the trees in the parking lot image were not in the original gray level image, and were completed by hand.

Figures 10c and 11c show ground level side views of the O-terminator hypotheses made automatically from the shadow information. Figures 10d and 11d show another view of both 3-D wire frame models. Note that for the objects in figure 10, only the building surfaces result from the correspondence between building boundaries and their shadows. The "trees" are the hypothesized illumination occluding profiles computed from the shadow shape. Note also that the O-terminator hypotheses for the vehicles in figure 11 are correctly interpreted as being off the ground. Also note that the breaks in the shadow boundaries of the cars do not correspond to breaks in the corresponding O-terminator boundaries. This indicates a change in the height of the O-terminator elements, which in fact correspond to the tops of the cars, and the container in the truck. The trees and the lamp post are the hypothesized illumination occluding profiles computed from the shadows.

6.5 CONCLUSION

One of the reasons that a human observer is able to obtain a strong impression of depth from a single monochromatic aerial image appears to be the observed shadows cast by the 3-D objects in the scene.

While other techniques such as shape from texture and shape from shading could determine whether the visible object surfaces are slanted or curved, they can not determine their height from the ground surface if they are elevated. We have demonstrated

that under simplifying assumptions, shadow analysis is useful to obtain 3-D information from the shadows cast by the objects in the scene, without prior knowledge of object position. This is particularly useful if we are interested in recognizing man-made objects, which have been a source of difficulty for techniques such as stereo.

We have also shown that our image segmentation techniques can provide stable line segment primitives suitable for the analysis of the shadows cast by the relatively small objects in the images shown. Our two shadow analysis techniques are suitable for an initial coarse segmentation of the image to locate the objects in the scene (by locating the shadows first and establishing partial object to shadow correspondences), and guide fine segmentation and analysis algorithms for detailed object descriptions and recognition.

We then come to the conclusion that the shadows cast become an important source of 3-D information information from a single monochromatic aerial image. Suggested extensions to this work include determining whether the two techniques discussed could cooperatively produce good results for images containing objects more complex than those shown here. With the region-based technique, it might prove useful to extract also other regions, with varying gray level intervals to locate the visible object surfaces as well as the shadow regions. More interesting, is to study how the information that can be derived from the shadows can be combined with other techniques, such as local shape from shading and stereo, to obtain more accurate and informed 3-D surface inferrencing techniques.

REFERENCES

[1] A. Huertas and R. Nevatia, "Detection of Buildings in Aerial Images Using Shape and Shadows," Proc. Int'l Joint Conf. on Artificial Intelligence, West Germany, August 1983.

[2] D. Waltz, "Understanding Line Drawings of Scenes with Shadows," The Psychology of Computer Vision (P. Winston ed.), McGraw-Hill, 1975, pp. 19-92.

[3] M. B. Clowes, "On Seeing Things," Machine Intelligence, Vol. 2, February 1971, pp.79-116.

[4] D. A. Huffman, "Impossible Objects as Nonsense Sentences," Machine Intelligence, Vol. 6, July 1971, pp. 295-323.

[5] S. Shafer and T. Kanade, "Using Shadows in finding Surface Orientations," Proc. DARPA Image Understanding Workshop, September 1982, pp. 90-102.

[6] D. Lowe and T. Binford, "The Interpretation of Geometric Structure form Image Boundaries," Proc. DARPA Image Understanding Workshop, September 1981, pp. 39-46.

[7] A. P. Witkin, "Intensity-Based Edge Classification," Proc. AAAI, Vol. 2, August 1982, pp. 36-41.

[8] T. O. Binford, "Inferring Surfaces from Images," Artificial Intelligence, Vol. 17, 1981, pp. 75-116.

[9] A. Huertas, "Classification of Edges for Object Detection in Aerial images," USC-ISG Technical Report, Vol. 102, October 1982, pp. 22-38.

[10] R. Nevatia and R. Babu,"Linear Feature Extraction and Description," Computer Graphics and Image Processing, Vol. 13, 1980, pp. 257-269.

[11] R. Ohlander, K. Price and R. Reddy, "Picture Segmentation Using a Recursive Region Splitting Method," Computer Graphics and Image Processing, Vol. 8, 1978, pp. 313-333.

[12] G. Medioni, "Obtaining 3D Information from Shadows in Aerial Images," Proc. CVPR, Washington D.C, June 1983, pp. 73-76.

[13] A. Huertas and R. Nevatia, "Edge Detection in Aerial Images Using $\nabla^2 G(x,y)$ Filters," USC-IPI Technical Report, Vol. 1010, September 1981, pp. 16-26.

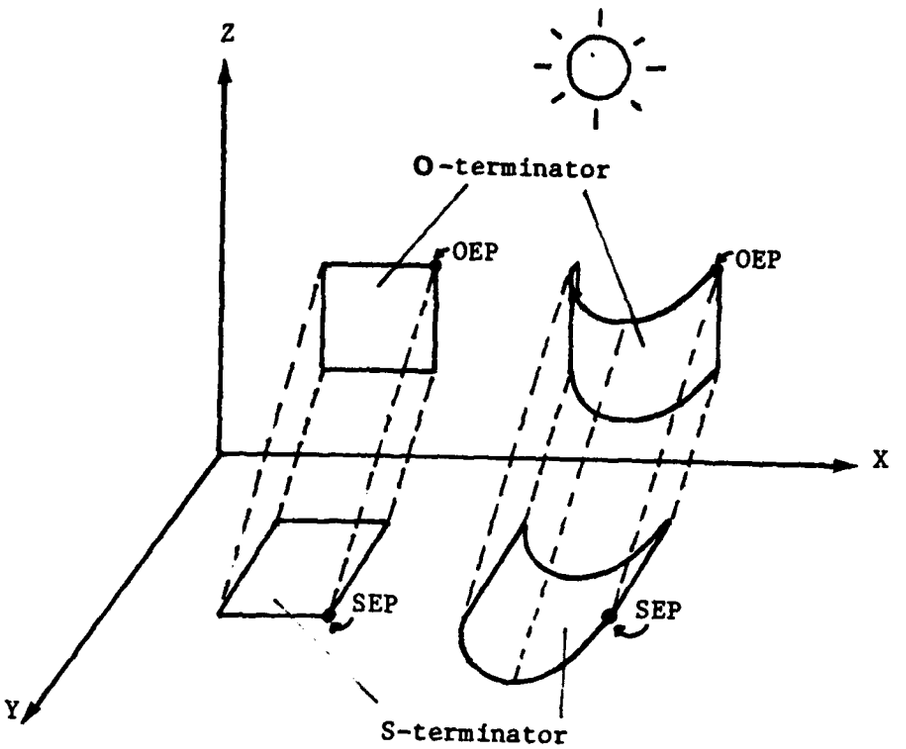


Figure 1: Shadow Solids

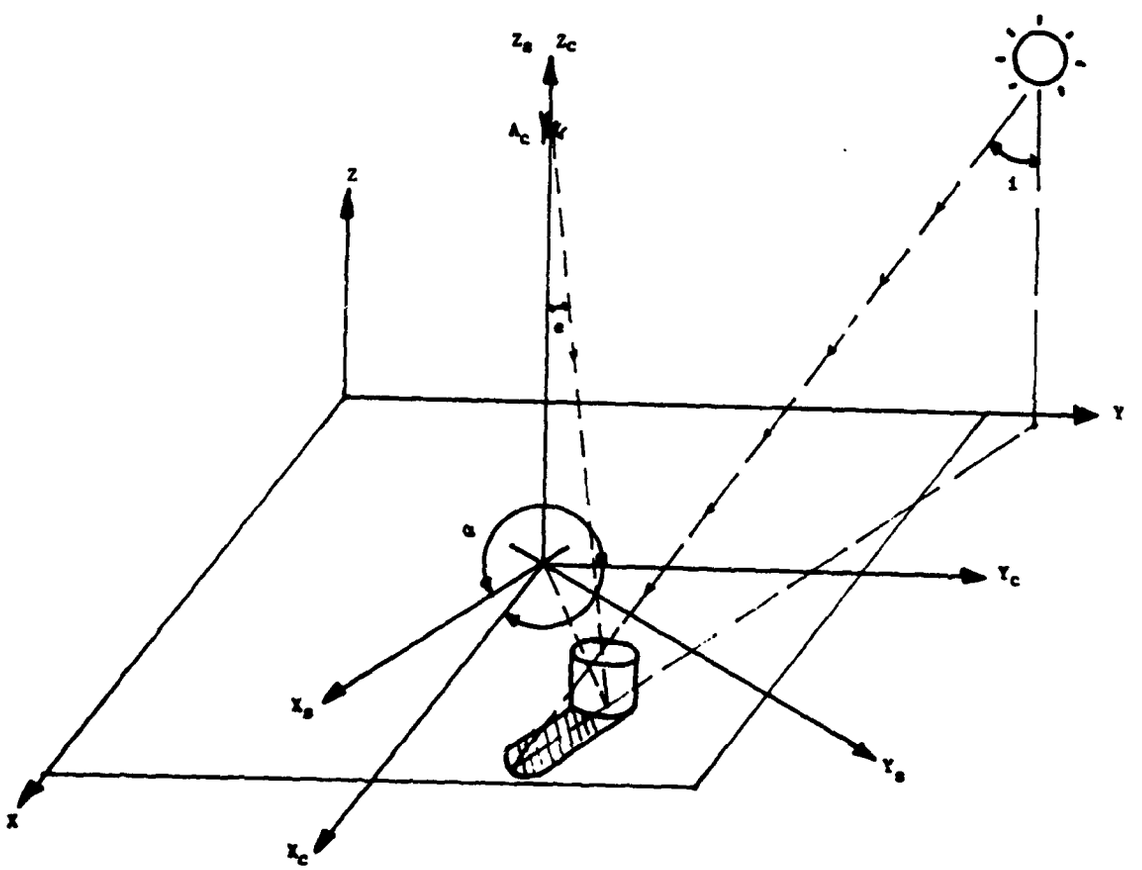
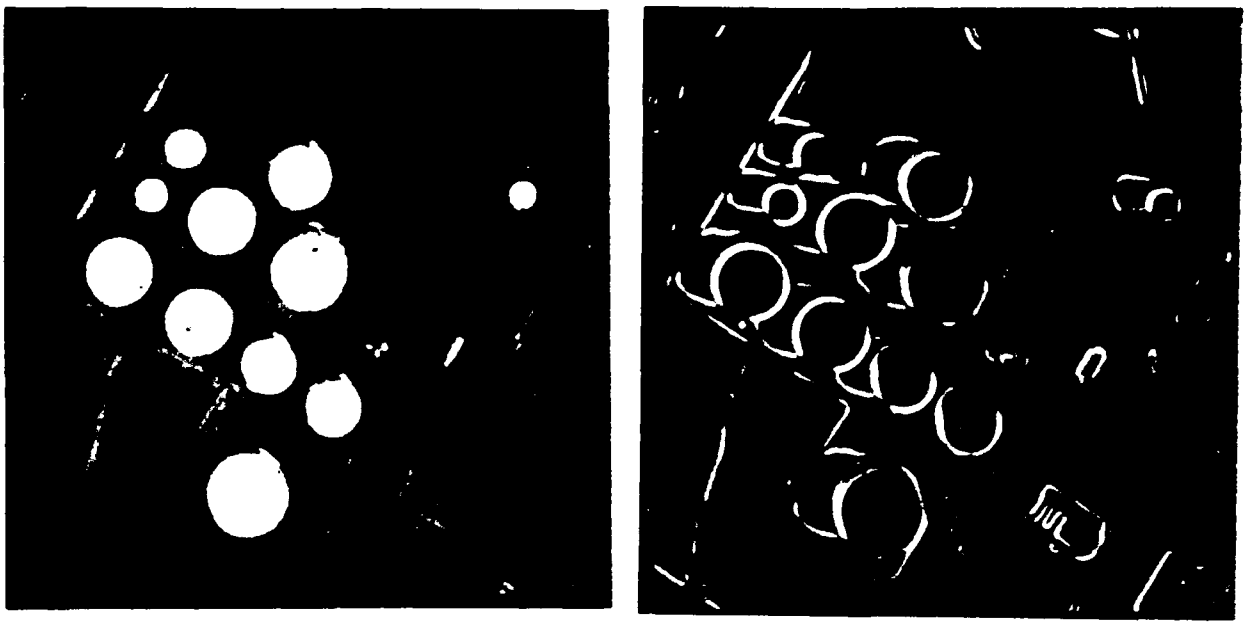
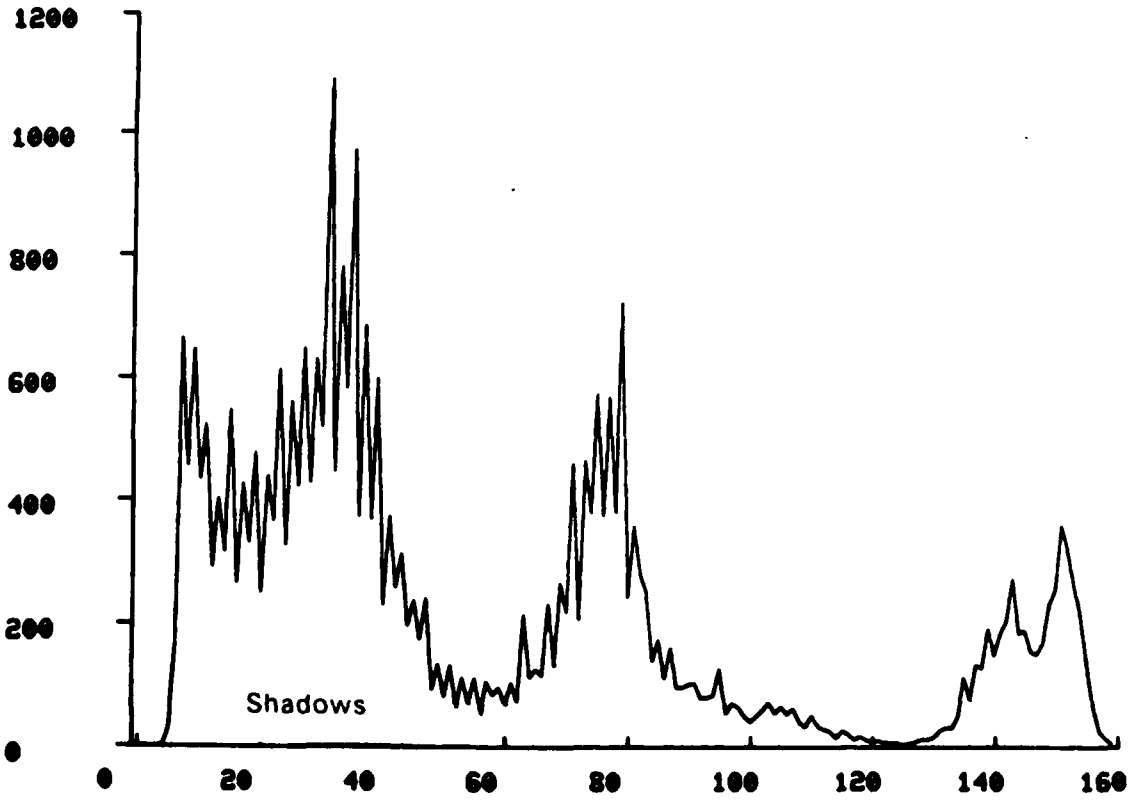


Figure 2: Shadow Geometry



a) Gray Level Image

b) Mask



c) Masked Histogram

Figure 3: Shadow Gray Level Range

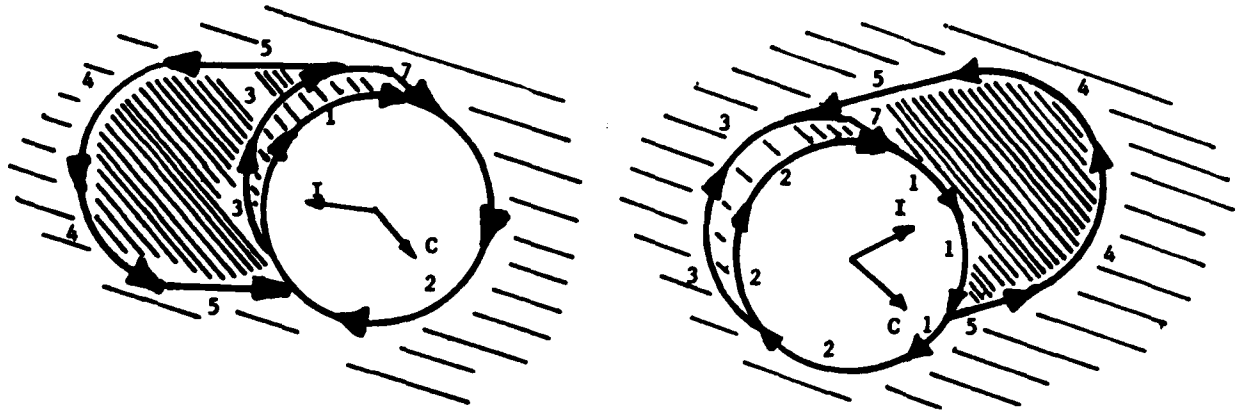
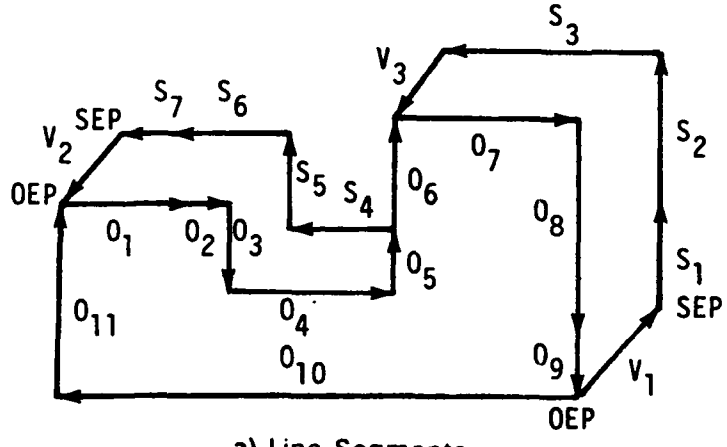
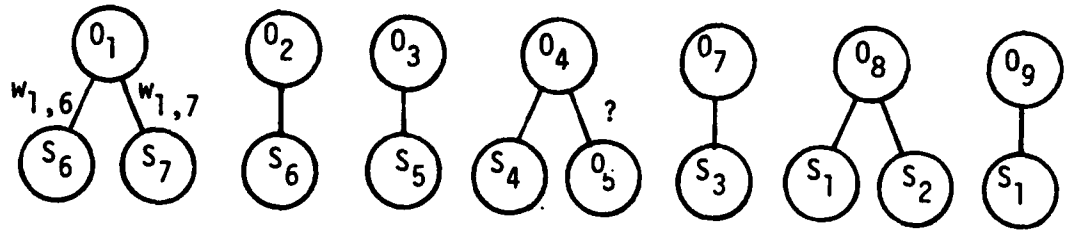


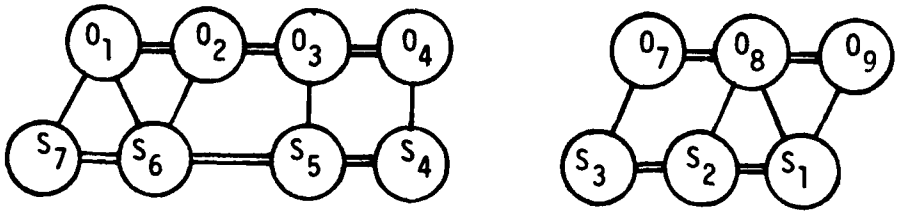
Figure 4: Line Segment Types



a) Line Segments

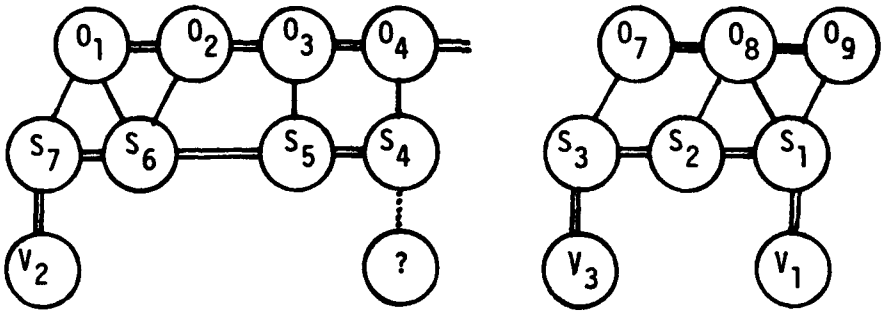


b) C-trees



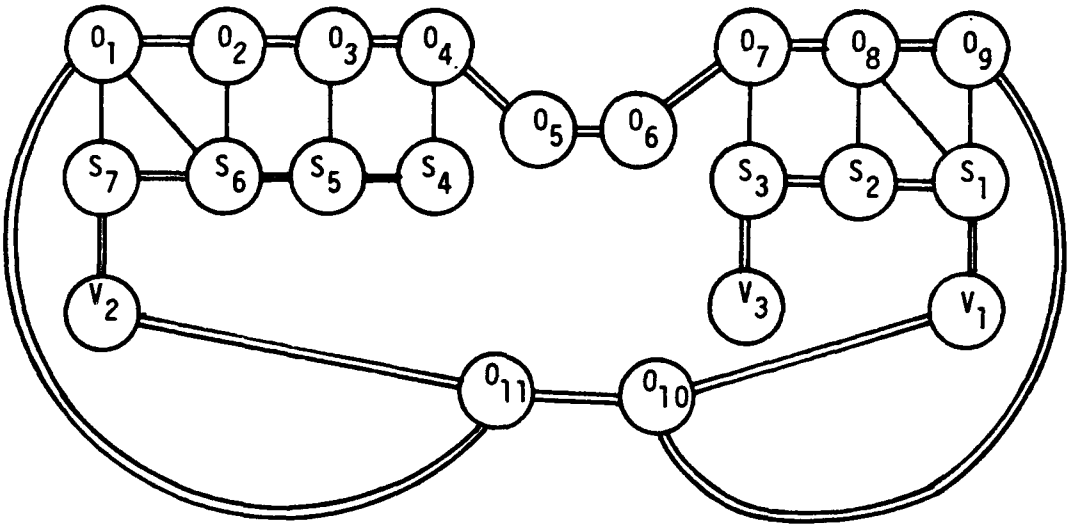
c) C-graphs

Figure 5: Object-to-Shadow Segment Correspondence



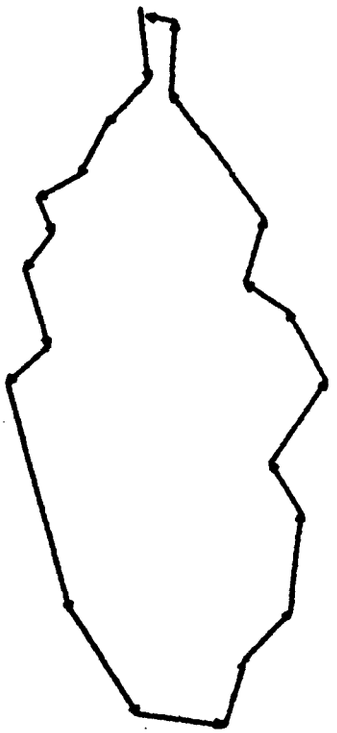
d) Augmented C-graphs

e) NC-chains

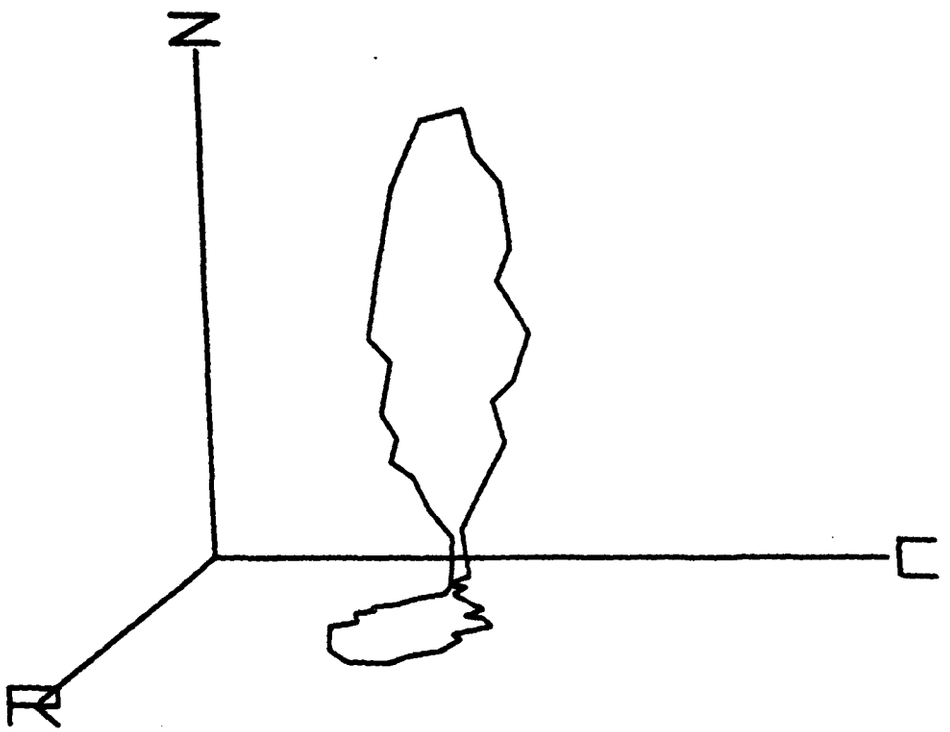


f) Complete C-graph

Figure 5: Object-to-Shadow Segment Correspondence (continued)



a) Shadow Boundary



b) Illumination Occluding Profile

Figure 6: Complex Object O-terminator Hypothesis

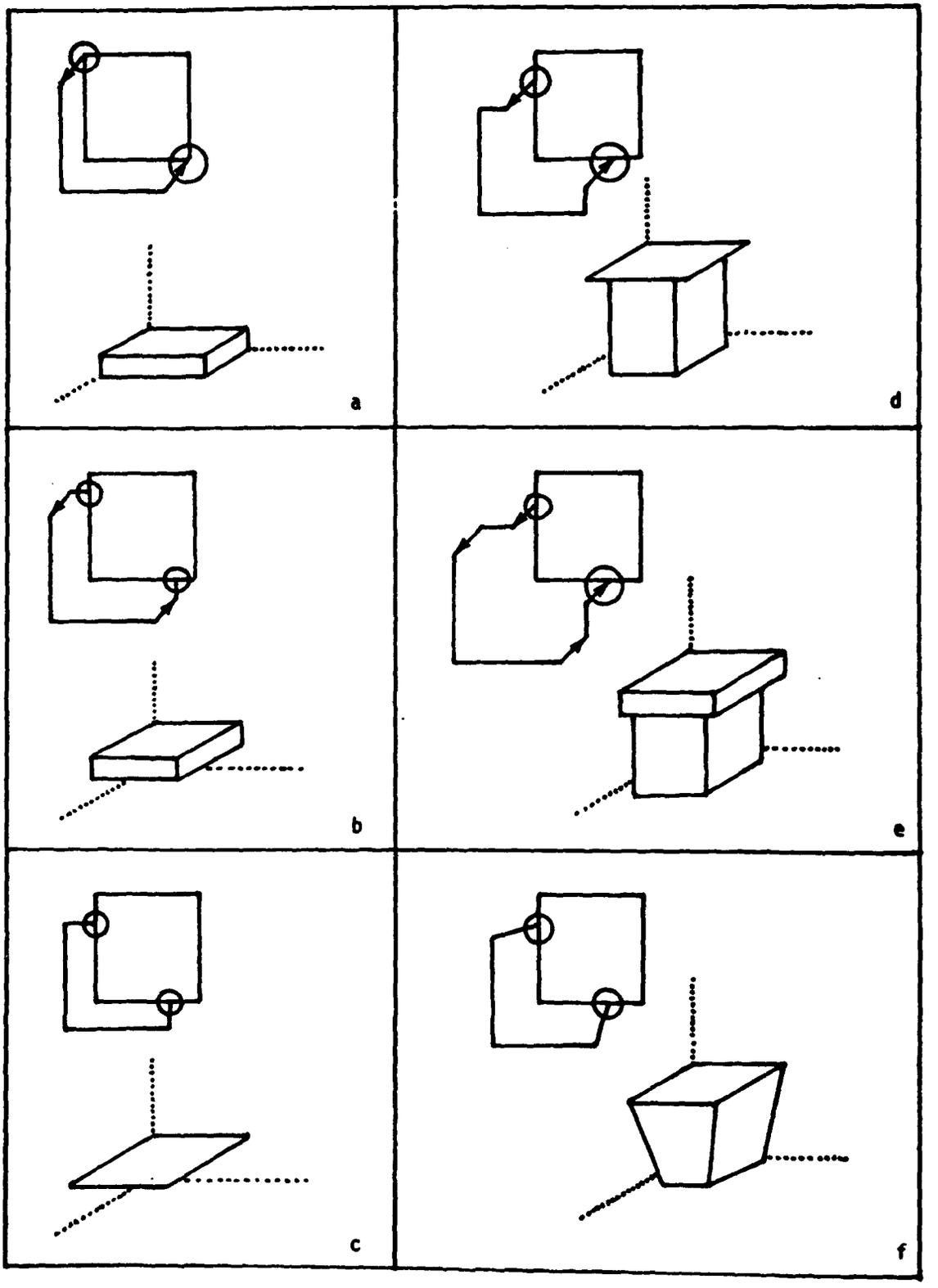


Figure 7: O-terminator Hypotheses (edge method)

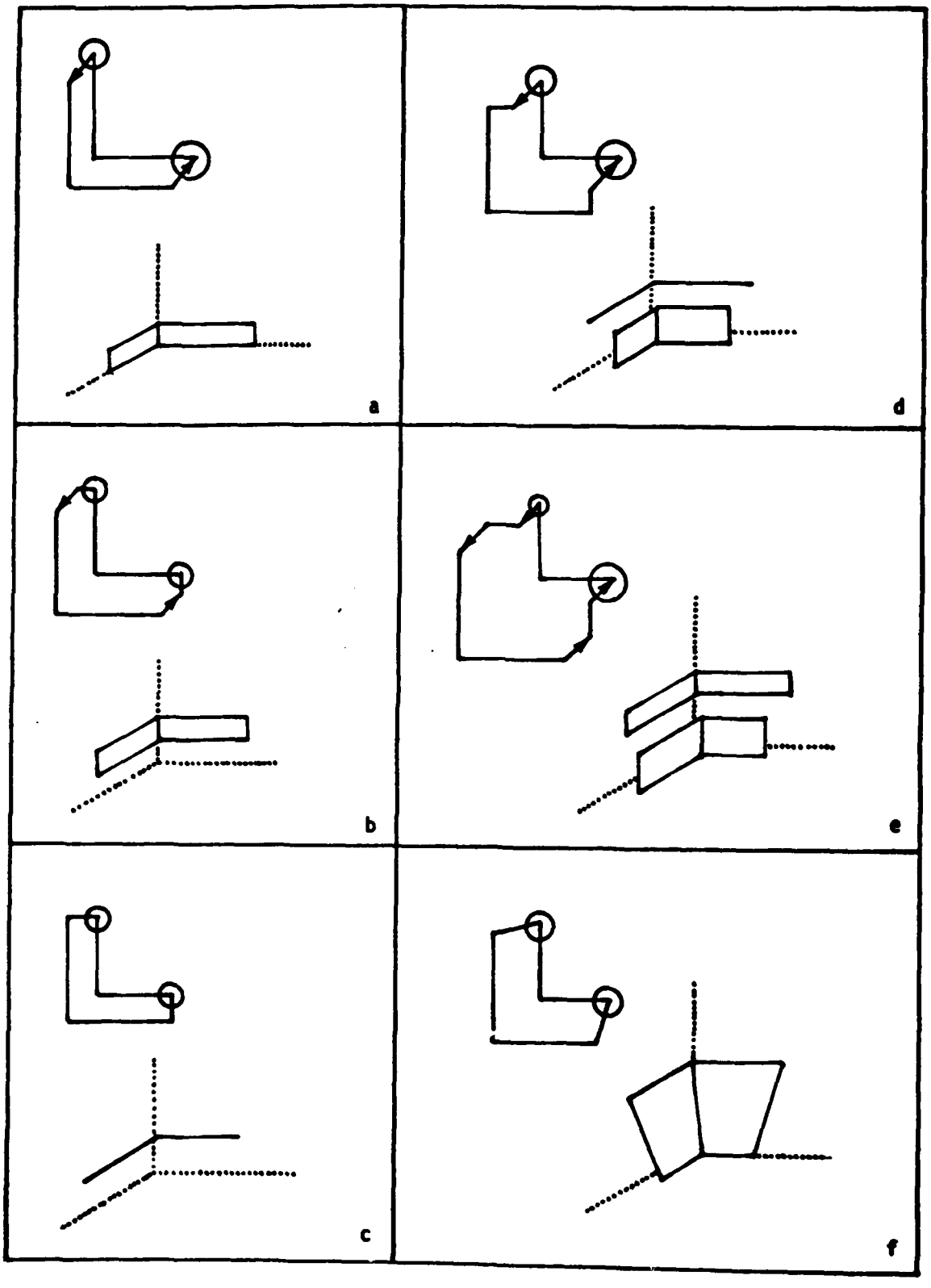
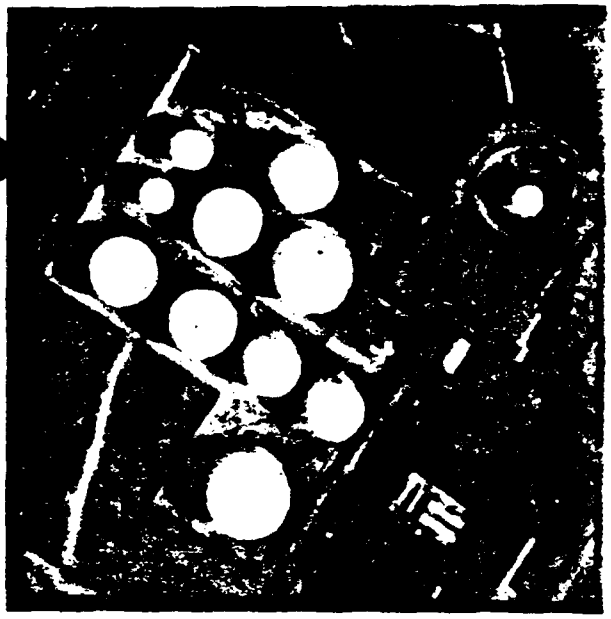
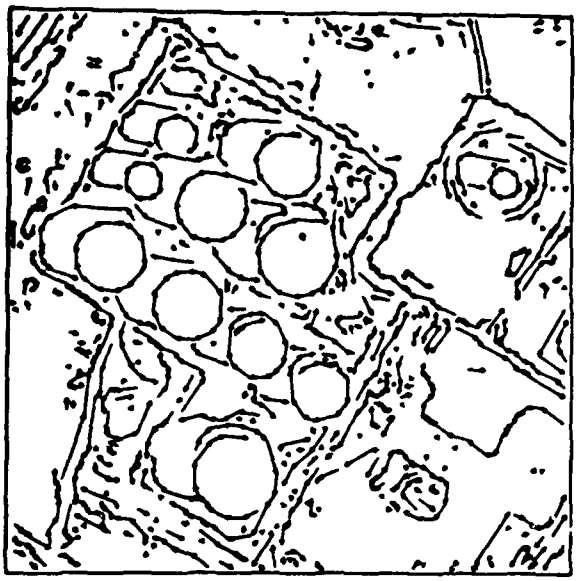


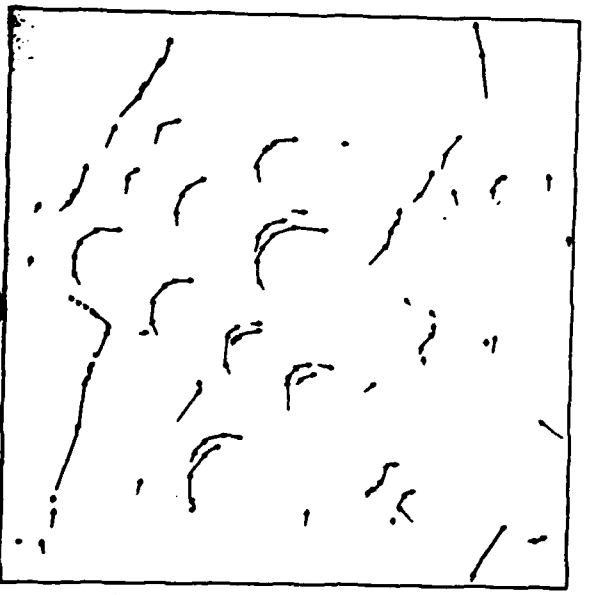
Figure 8: O-terminator Hypotheses (region method)



a) Gray Level Image

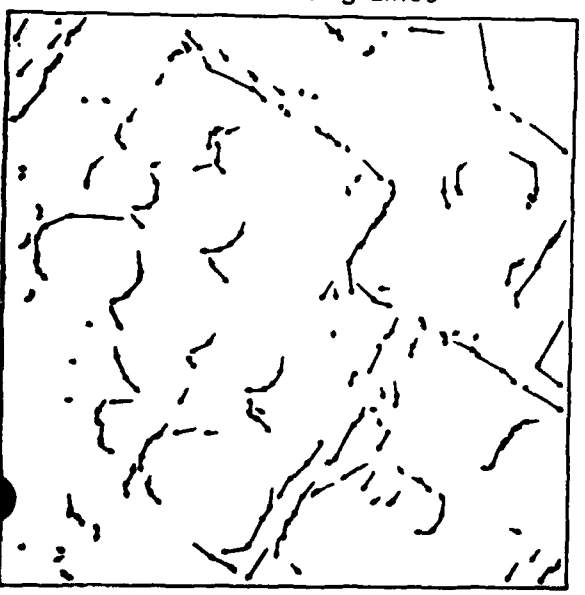


b) Line Segments

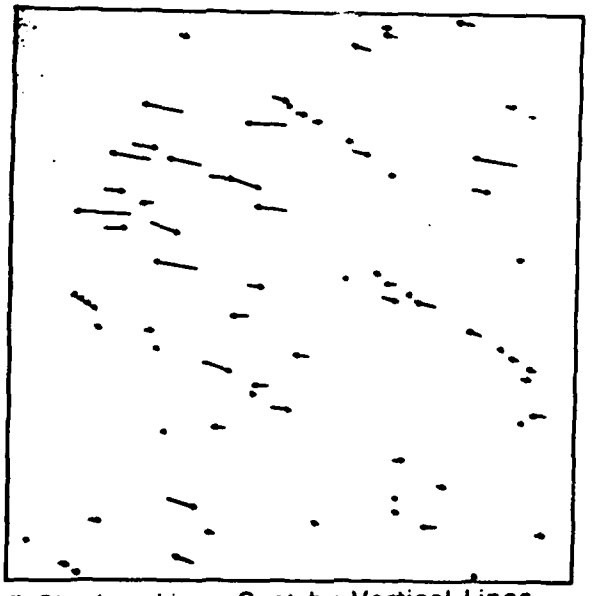


c) Shadow Casting Lines

d) Non-shadow Casting Lines

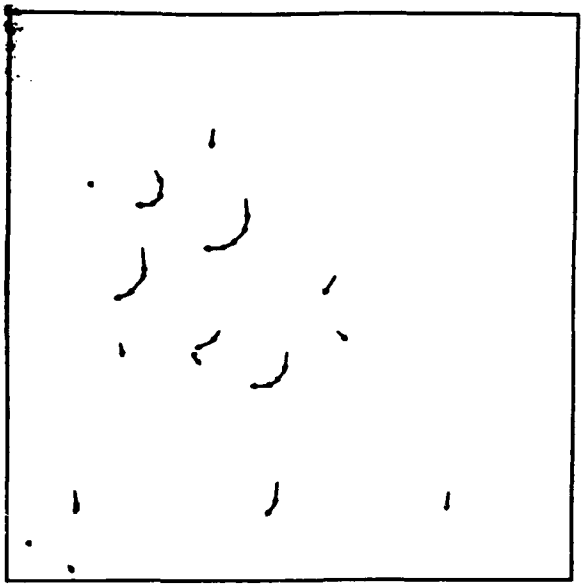


e) Shadow Lines

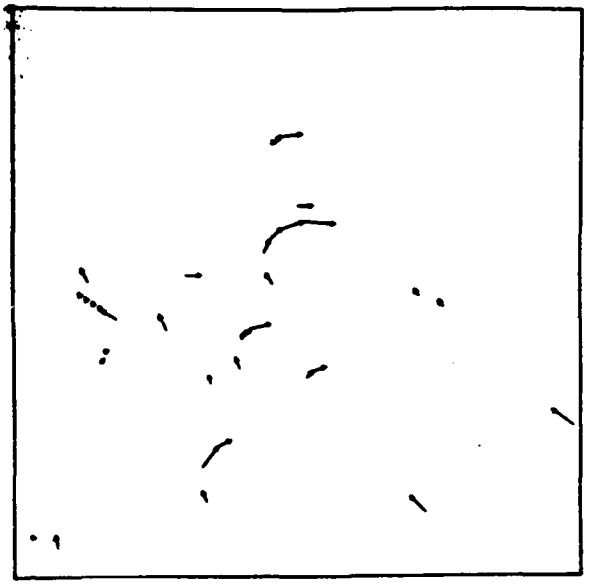


f) Shadow Lines Cast by Vertical Lines

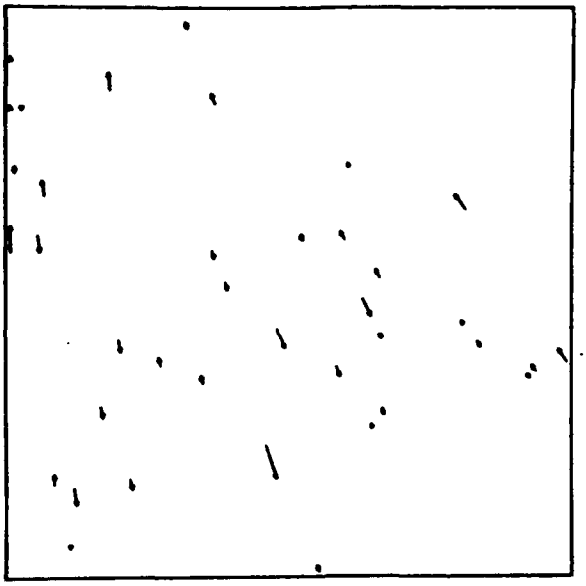
Figure 9: Results for Tanks Image



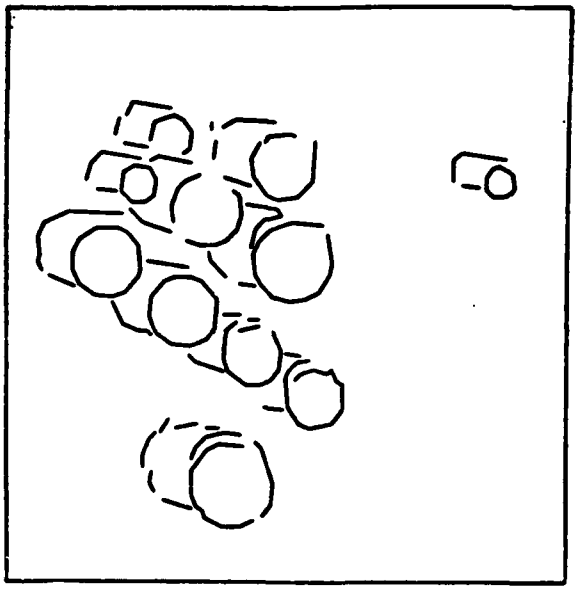
g) Shadow Occluding Lines



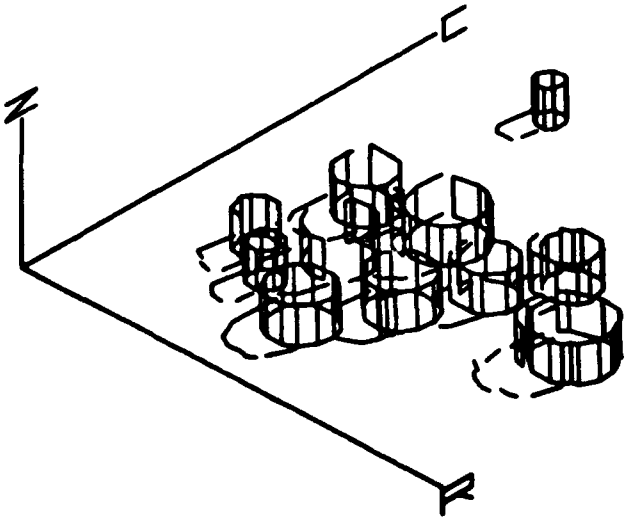
h) Lines not on Top Surface



i) Vertical Linas



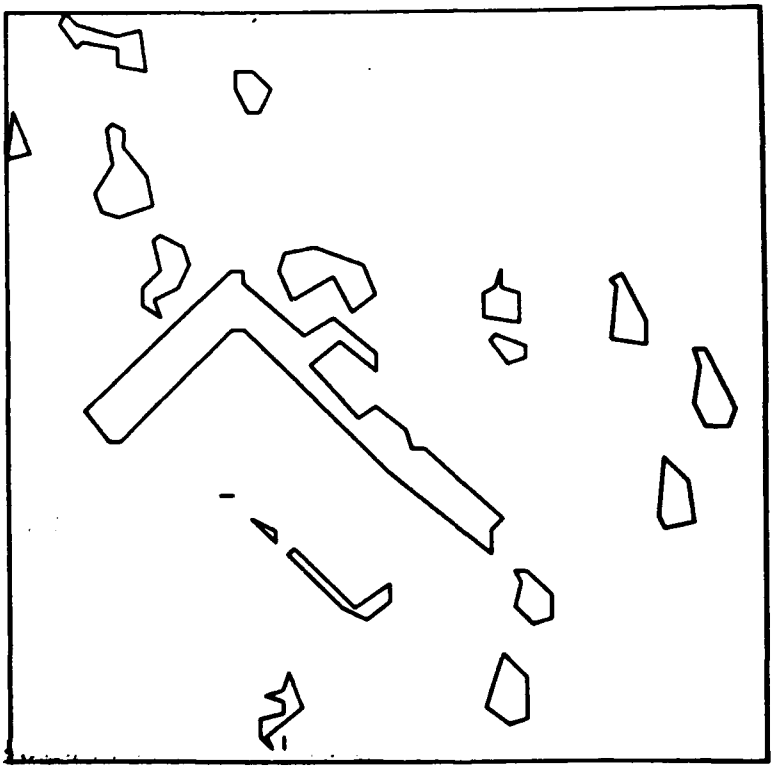
j) Object and Shadow Outlines



k) 3-D Wire Frame Model

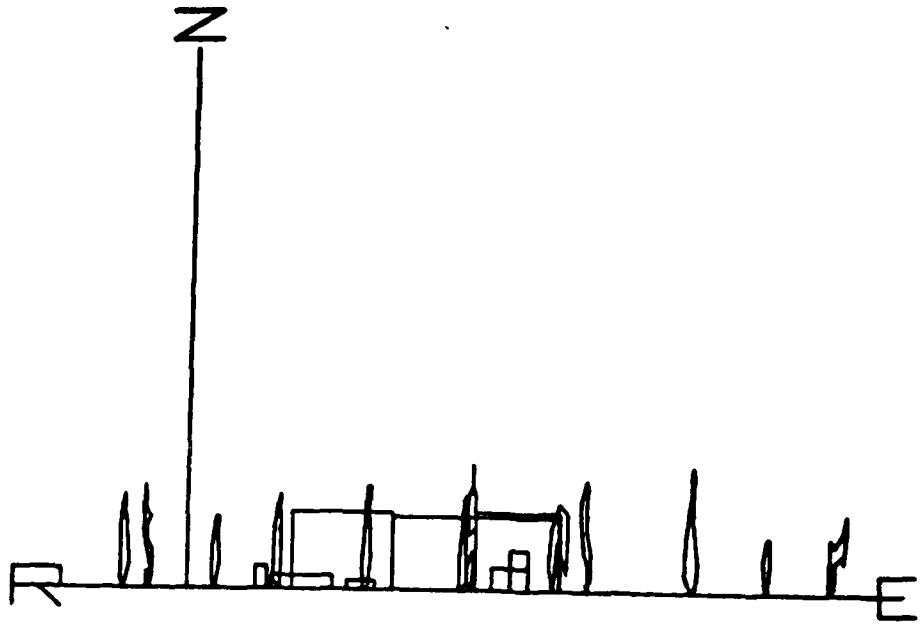
Figure 9: Results for Tanks Image (continued)

a) Gray Level Image

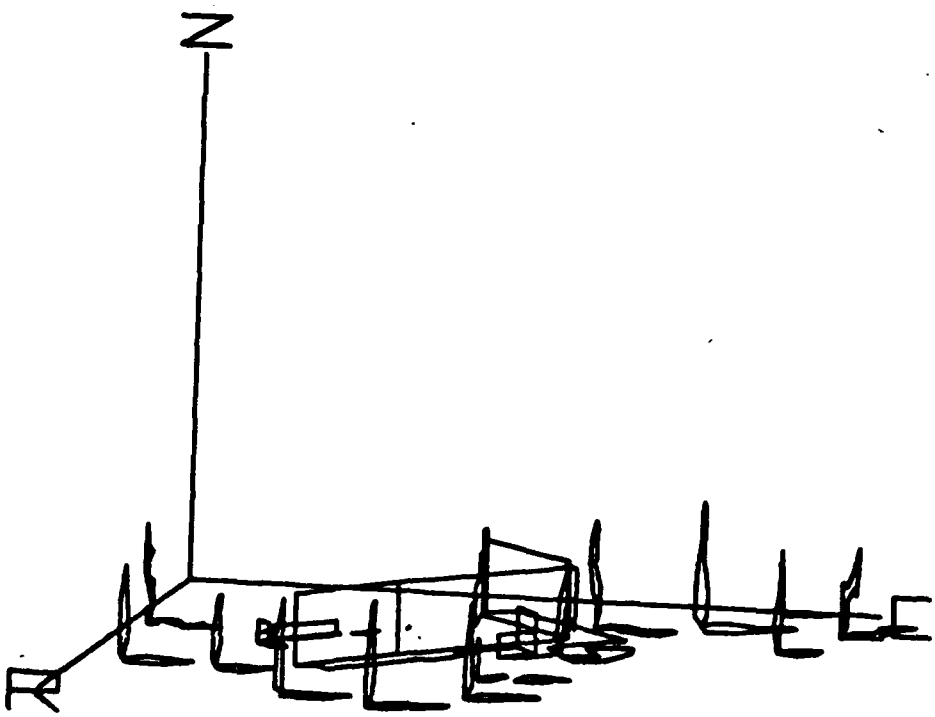


b) Shadow Regions

Figure 10: Results for Building Image

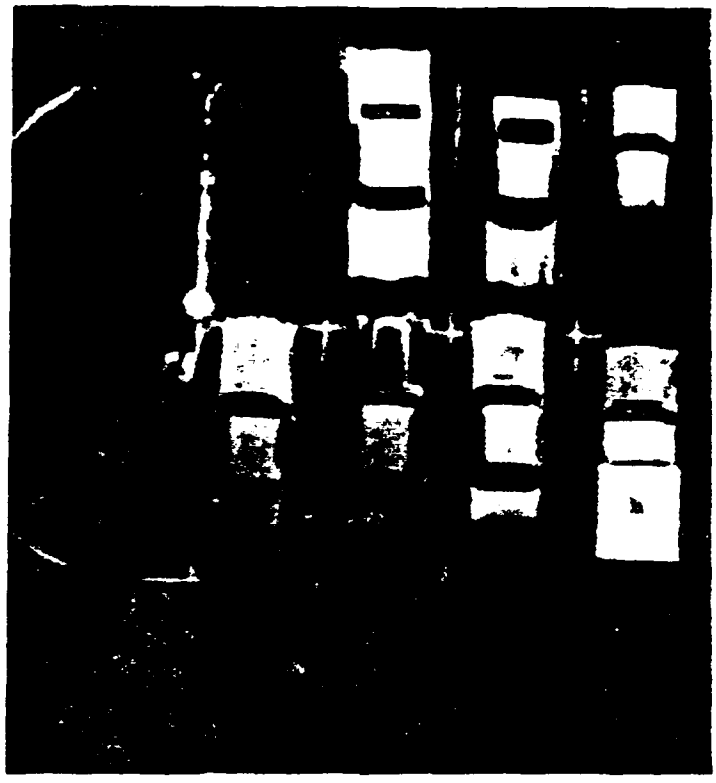


c) Side View of 3-D Wire Frame Model

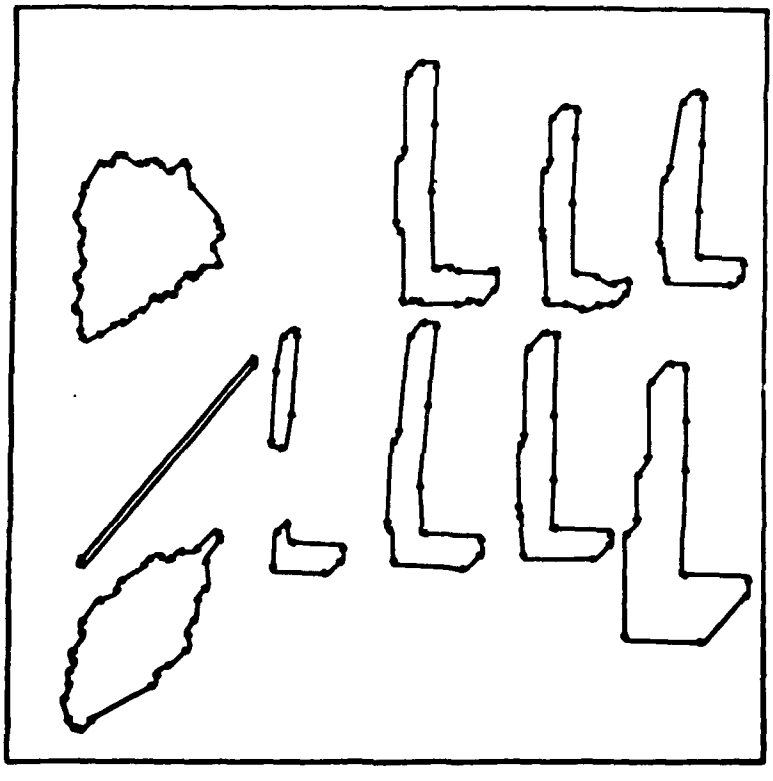


d) Rotated View of 3-D wire Frame Model

Figure 10: Results for Building Image (continued)

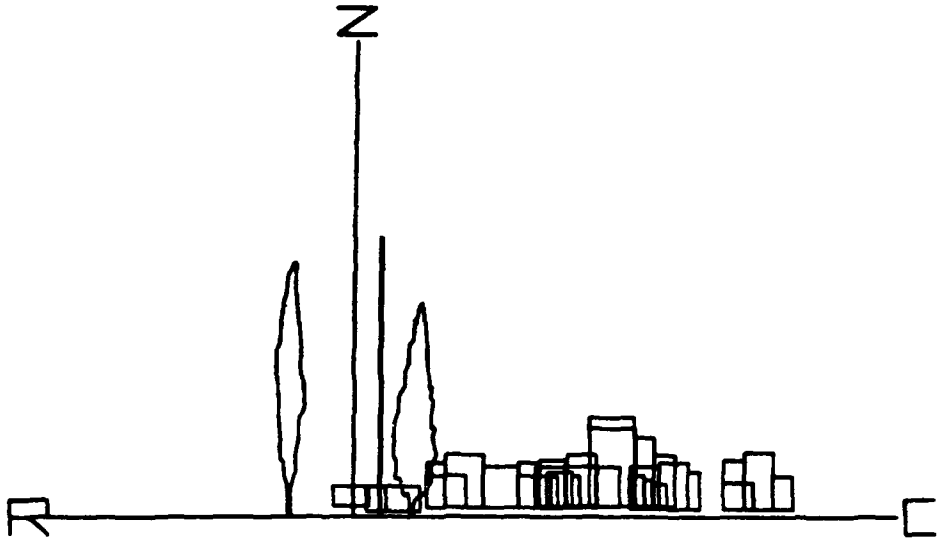


a) Gray Level Image

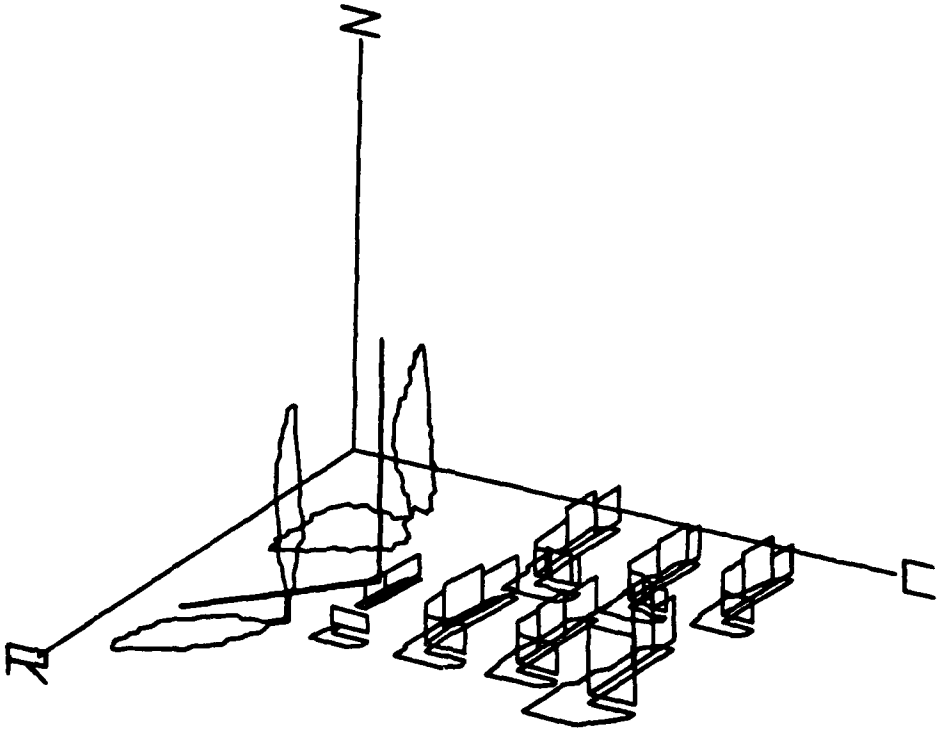


b) Shadow Regions

Figure 11: Results for parking lot image



c) Side View of 3-D Wire Frame Model



d) Rotated View of 3-D wire Frame Model

Figure 11: Results for Parking Lot Image (continued)

SECTION 7

IMPLEMENTATION OF SHAPE-FROM-SHADING ALGORITHMS

Sheng Hsuan Yu

7.1 INTRODUCTION

One purpose of a visual system is to reconstruct a 3-D model of the world from a 2-D image. When we look at an image, we have no problem in identifying the objects and their shapes no matter how complicated they are. Our visual system is very robust in perceiving the shape of a surface; but we know little about it.

One method for depth perception that is relatively well understood is binocular stereopsis. In stereo computation, object features in the left and right view are placed in correspondence. The difference in the projection of the corresponding points in two views is used to determined the depth of the surface along that contour. Since the stereo computation can determine depth only at particular points in the image, we have to use other cues to recover the complete surface of the whole regions. An interpolation method can be used to compute a complete description. However, this may give a poor description, when the contour information is sparse.

When we look at a single image, we still can find other cues to help us perceive the shape of the surface of an object. One of these cues is shading. The ordinary visual world is mostly composed of opaque 3-D objects. The intensity of a pixel in a digital image is produced by the light reflected by a small area of the surface near the corresponding point of the object. Different surface normals will produce different image intensities. The changes of image intensities in a smooth region give us information about the surface shape.

Many shape-from-shading techniques have been proposed. Woodham [1] uses multiple images to obtain a solution. A global relaxation method in [2] has been used to recover shape by propagating constrains from boundary conditions (such as surface normals from smooth occluding contours) over the surface whose shape is to be estimated. Pentland [3] has formulated a method for deriving surface shape from local image shading. It utilizes local changes only in image intensity to estimate shape and doesn't require a priori knowledge about the viewed scene. It assumes that albedo and illumination are constant in the neighborhood of the point being examined, the surface reflects light isotropically, and the surface principle curvatures are equal and nonzero.

In this report, we carefully evaluate the two methods mentioned above by applying them to both synthetic and natural images. We also examine the possibility of combining these methods, that is, to use the output of the local estimation method to bootstrap the global method.

7.2 REPRESENTATION

7.2.1 Gradient Space

There are various ways to specify the surface orientation of a plane. One can use the equation defining the plane or the direction of a vector perpendicular to the plane. If the equation of a plane is ax+by+cz+d=0, the surface normal is (a,b,c). This method is easily extended to curved surface by applying it to tangent planes. If the equation of a surface is given explicitly as:

$$-z=h(x,y),$$

then the surface is given by the vector (p,q,-1), where

$$p=-\partial z/\partial x, \quad q=-\partial z/\partial y$$

The quantity (p,q) is called the gradient of h(x,y) and the gradient space is the 2-D space in which coordinates are p and q. Geometrically, we can think of this as the projection of the "Gaussian sphere" on a plane tangent to its north pole. The center of the projection is the sphere center.

7.2.2 Stereographic Space

As we shall see, points on the equator of the Gaussian sphere correspond to surface patches on the occluding boundary. With the gradient space projection, the equator maps to infinity. As a result, a point on occluding boundaries, which plays a very important role in the global relaxation method, is undefined in gradient space. We can consider sufficient large values of p and q to be approximation of the gradients of points on the equator of the Gaussian sphere. The other solution is to use the stereographic projection. Two axes of stereographic space are labeled by f and g and they can be converted from pq-space directly [2]:

$$f=2p(\sqrt{1+p^2+q^2}\ -1)/(p^2+q^2)$$

$$g=2q(\sqrt{1+p^2+q^2}\ -1)/(p^2+q^2)$$

The center of the projection is the south pole instead of the center of the sphere. The use of stereographic space instead of gradient space doesn't change the constrains that restrict the surface orientations.

7.2.3 Slant and Tilt

We also use slant and tilt representation. It is a modified representation of a point (p,q) in the gradient space by using the polar coordinates τ and tanσ. They can be converted from pq-space by the following formulas:

$$\tau = \arctan(q/p)$$

$$\tan\sigma = \sqrt{p^2+q^2}$$

The tilt specifies the direction in which the surface normal is oriented and the slant is the angle between the surface normal and the viewer.

7.3 SHAPE FROM SHADING

In this section, we briefly outlines the algorithms for each method and the computation techniques.

7.3.1 Global Relaxation Method

Our work is based on [2]. The global relaxation method uses the image-irradiance equations and smoothness criterion as constrains. It has been shown that the recovery of surface orientations is possible if the position of light source and the object reflectivity are known. Let f and g be the stereographic coordinates. We define some error terms: $s_{i,j}$ is defined as the departure from the smoothness, $r_{i,j}$ is defined as the difference of the image intensity and the value computed from reflectivity function by putting current $(f_{i,j},g_{i,j})$ into it.

$$s_{i,j} = [(f_{i+1,j}-f_{i,j})^2+(f_{i,j+1}-f_{i,j})^2+(g_{i+1,j}-g_{i,j})^2+(g_{i,j+1}-g_{i,j})^2]/4$$

$$r_{i,j} = (I_{i,j}-R(f_{i,j},g_{i,j}))^2$$

The total error E is defined as follows:

$$E = \sum_i\sum_j(s_{i,j}+\lambda r_{i,j})$$

where λ weights the relative importance between smoothness and reflectance errors.

Given adequate initial conditions, the goal is to find out the surface normals elsewhere on the surface that minimize the total error, E. To obtain this goal, Gauss-Seidel like iteration method is derived as follows:

$$f^{n+1}_{i,j}=f_{av}{}^{n}_{i,j}+\lambda(I_{i,j}-R(f_{av}{}^{n}_{i,j},g_{av}{}^{n}_{i,j}))\partial R(f_{av}{}^{n}_{i,j},g_{av}{}^{n}_{i,j})/\partial f$$

$$g^{n+1}_{i,j}=g_{av}{}^{n}_{i,j}+\lambda(I_{i,j}-R(f_{av}{}^{n}_{i,j},g_{av}{}^{n}_{i,j}))\partial R(f_{av}{}^{n}_{i,j},g_{av}{}^{n}_{i,j})/\partial g \qquad (1)$$

where f_{av} and g_{av} are the averages of their four neighboring points.

The iteration ends when E is sufficiently small or reaches a stable value. This method needs a lot of a priori knowledge about the image, to reduce convergence time. In the following paragraphs, we will discuss how to choose or estimate the boundary conditions.

- Occluding boundary. Since the surface normal of the point on the silhouette is per-

pendicular to the line of sight and the tangent line of the silhouette, it can be determined uniquely. A smooth shading region is a region which has a wide range of the intensity distribution and has no occluding edge within it. By using a region growing method, we segment such a region into many elongated(strip) regions with long artificial boundaries. If the boundary of such a region matches with the edge from zero-crossing, it is very likely to be occluding boundary.

- Local estimates. Use the output from the local estimates that will be discussed in the next section as the starting points.

7.3.2 Local Estimation

Pentland formulated a method that utilizes only local changes in image intensity to estimate shape without a priori knowledge about the image. He proposed two estimators for slant and tilt of a surface normal by assuming that the surface is a Lambertian surface and the surface principle curvatures are equal and nonzero.

1. Tilt estimator: The tilt of the surface is the image direction in which the second directive of image intensity d^2I is greatest.
2. Slant estimator: The arccosine of the surface slant , Zn, the z component of the surface normal, is estimated by the following equation:

$$Zn = -c \,/\, \sqrt{(|\nabla^2 I / I| - c^2)} \qquad (2)$$

where c is a constant related to the surface curvature, and

$$\nabla^2 I = \partial^2 I/\partial x^2 + \partial^2 I/\partial y^2.$$

In digital images, the Laplacian is usually approximated by [4],

$$\nabla^2 I(i,j) = I(i+1,j)+I(i-1,j)+I(i,j-1)+I(i,j+1)- 4I(i,j).$$

Since very accurate values of $\nabla^2 I$ are required in the above estimators, this function doesn't work here. Another way is to approximate $\nabla^2 I$ by convolving the operator $\nabla^2 G$ with the image. This operator is determined by two parameters $\omega(=2\sigma$,standard deviation of G) , the width of the central excitatory region, and $\lambda(=2\pi\sigma)$, the diameter of the operator. This is a circularly-symmetric filter as shown in figure 1.

There are two reasons for using $\nabla^2 G$ operator. First, it incorporates a smoothing function that makes it less sensitive to noise and quantization. Second, the operator covers larger areas and takes more elements into consideration so that the output is more stable and accurate. The size of the operator affects the quality of results. The details will be discussed in section 5. Since albedo and illumination are assumed to be locally constant, dividing $\nabla^2 I$ by I yields a measure that depends primarily on the surface curvature and the slant. A good estimate of constant c within an image region is obtained by applying the constraint that $-1 \leq Zn \leq 0$, which means a visible surface facing the viewer.

$$c=\sqrt{\min(|\nabla^2 I/I|)/2} \quad (3)$$

Then, the z component of the surface normal can be obtained.

Estimating the tilt from the slant estimates: To compute the tilt is to find out the direction **k**(=<u,v>) along which d^2I is maximum. The second derivative of I along a vector **k** is given by

$$d^2I = I_{xx} \cdot u^2 + 2I_{xy} \cdot uv + I_{yy} \cdot v^2.$$

where $I_{xx}=\partial^2 I/\partial x^2$, $I_{yy}=\partial^2 I/\partial y^2$ and $I_{xy}=\partial^2 I/\partial x \partial y$.

Thus directional second directive along a direction **k** can be calculated by using values of Ixx, Ixy and Iyy. But very accurate values of them are required and it is not computationally efficient because separate operators are needed for each of them. Pentland [3] suggested a method to approximate the directional second directive operator by summing the values of $\nabla^2 I$ along a straight line. Although this method is more convenient than the previous one, it still requires a lot of efforts to compute the sums along many different lines and to normalize them so that they can be compared. Intuitively, the direction along which the slants have the biggest changes is the direction in which the surface is oriented. So the tilt can be approximated by

$$\tau = \arctan(dy/dx),$$

where dx and dy are the first-order differential derivatives in x and y axis.

We have found that it outperforms the previous methods and is very attractive computationally.

7.4 INTEGRATION ALGORITHM

Gradient space, stereographic space ,and the slant-and-tilt representation are all good representation of the surface normals, but not convenient for display. Integration of these representations of surface normals into a depth map makes it easier for us to judge how accurate of the recovered shape is.

p and q are partial derivatives of z along x and y axis, and hence give changes of z in x and y direction, respectively. Suppose that the depth at point (x0,y0) is known as z0. Let x1=x0+dx and y1=y0+dy and z1=f(x1,y1). Then z1 can be approximated by the following equation:

$$z1 = z0 + (1/2)dx(p(x0,y0)+p(x1,y1))+(1/2)dy(q(x0,y0) + q(x1,y1)) \quad (4)$$

The integration procedure includes the following steps.

1. Choose the starting point, (x0,y0), and assign z0 a value. For example, the center of

a sphere is a good starting point.
2. Integrate the depth along an axis, i.e., x-axis or y-axis.
3. Based on the depth of the axis chosen in the last step , continue the integration process toward the two opposite sides normal to this axis.

If the surface normal is represented by the slant and the tilt components, a transformation is needed, $p=\tan\sigma\cos\tau$ and $q=\tan\sigma\sin\tau$. In the next section, we use this scheme to transform the orientation representation into a depth map, which proves to be very helpful for our perception of the recovered shape.

7.5 RESULTS AND DISCUSSION

We have applied these shape-from-shading techniques to several synthetic and natural images. In local analysis, we used a synthetic image of 64x64 pixels as shown in figure 2(a) to test the effect of the size of the operator on the recovered shape. The operators used are the same as the Laplacian-Gaussian filter for local edge detection discussed in [5]. Figure 2 (c)-(f) show the depth maps recovered from different size $\nabla^2 G$ operators with $\omega=1,3,3,5$ and $\lambda=3,7,11,15$, respectively. Compared to the true depth shown in figure 2(b), the results become better as the size of the operators increase. It shows that the operator with $\lambda=15$ is quite good for an image of 64x64 pixels. Therefore, the operators used in the following experiments are around this size.

Figure 3(a) shows the synthetic images of Lambertian ellipsoid and hyperboloid and both are 64x64 images . Their true depth maps are shown in figure 3(b). These samples consist of different types of surface with different combinations of principle curvatures. Using Pentland's local estimation method, the recovered shapes of these images are shown in figure 3(c). Compared to the true shapes, the results are quite accurate except at the points along the boundaries and at points on the valley(saddle) of the hyperboloid.

If the global relaxation method is used to recover shape, we need some a priori knowledge about the surface, i.e., occluding boundaries and the illumination angle. Figure 4(a) shows the occluding boundaries of the synthetic sphere and hyperboloid as shown in the previous figures. The points on these boundaries provide the initial conditions for the global relaxation method. The sphere has a closed occluding boundary, while the hyperboloid has two separate occluding boundaries. The recovered shapes are given in figure 4(b). Both are very accurate compared to the true shapes given in previous figures. It shows that a closed occluding boundary is not necessary.

We also have experimented on a synthetic image with a reflection function other than Lambertian surface. The brightness of the surface depends linearly on the incident angle rather than the cosine of the incident angle.

$$I \propto 1-2i/\pi$$

The generated image is shown in figure 5(a). We still perceive it as a sphere. The recovered shape by local estimation method is shown in figure 5(b). The estimations around high-slant area are quite accurate. It is interesting to know that there is a dip in the center, which is a low-slant area. This is because the estimators are based on the as-

sumption of Lambertian surface. So the estimators are very sensitive to the changes of brightness when the incident angles are very small (the curve of cosine is relatively flat here). The global relaxation method has also been applied to this image and the recovered shape more resembles the true shape as the result shown in [2]. Figure 6 shows the profiles of the true shape and the shapes by local analysis and the global relaxation method.

Figure 7(a) is the digital image of "Lenna". We have a very strong perception of shapes in this image because of rich shading information, such as the part of shoulder. We used it as a test case for both local estimation and global relaxation methods.

Local analysis method is not suitable for a relatively flat area because the output of the Laplacian-Gaussian convolver on the area is very small. As shown in figure 7(b), the background has a very strange recovered shape, not a plane as we perceive it. After removing the background, the result looks better as shown in figure 7(c). For a surface of complex shape, local analysis will cause errors. This is because c value can not be estimated correctly. It is better to split a complex shape into small homogeneous regions.

The segmentation of this image by region growing is shown in figure 7(d). It shows that the intensities of these segmented regions change gradually from the southwest corner to the northeast corner, but there is an abrupt change near the boundary. So this boundary can be regarded as an occluding boundary as shown in figure 7(e). These initial conditions are chosen from the output of the pentland method. By giving a guessed orientation of the light source based on the direction along which the changes of intensities occur in the segmented regions, the recovered shape of the shoulder by relaxation method is shown in figure 7(f).

Figure 8(a) shows part of the face of "Elaine". As shown in Figure 8(b), the shape of the nose is correctly recovered by local estimation method. But the shape around right cheek is not. As we see clearly the black part of the nose, it seems to be illuminated indirectly or looks like a self-shadowed area. This may confuse the shape-from-shading algorithms. Noisy spots may cause errors as well as the area around zero-crossings. Because no clear occluding boundary can be found, it is very difficult to apply the global relaxation method. Using the result in figure 8(b) as initial conditions and these conditions are changeable, it reaches a solution after many iterations(400). Again, the orientation of illumination is guessed. The result is shown in figure 8(c).

7.6 CONCLUSION

We have applied shape-from-shading algorithms to several synthetic images and natural images. They both work well on synthetic images as illustrated in the last section, but the performance for natural images is poor. The reasons for this are not completely clear and require further investigation.

Some possible causes for the errors are:

a) Inadequate resolution
b) Non-Lambertian surface
c) Complex surface curvature (e.g. non-spherical)
d) Local changes in albedo
e) Complex illumination, including reflections from other objects and multiple light sources.

Pentland's method seems to be tolerant to non-spherical shapes, as seen by examples of synthetic surfaces with different curvature, but in natural images, it is effective only in some areas of high curvature, e.g. a nose. The method is sensitive to the choice filter size, which is hard to determine for a natural image. The method factors out albedo changes, but the effect of local changes, as in texture, is unclear.

The global relaxation method requires a priori knowledge of light source and reflectance function, which are not generally available for natural images - it is also difficult to obtain the needed initial conditions.

Our experiments indicate that shape from shading algorithms require further investigation before they can be applied to complex, natural images.

REFERENCES

[1] R. J. Woodham, "Analysis of Images of Curved Surfaces", Artificial Intelligence, August, 1981.

[2] K. Ikeuchi and B.k.P. Horn, "Numerical Shape From Shading and Occluding Boundaries", Artificial Intelligence 17, 1981.

[3] A. P. Pentland, "Local Shading Analysis", Technical Report 272, SRI International Artificial Intelligence Center, November, 1982.

[4] A. Rosenfeld and A. C. Kak, "Digital Picture Processing", Academics Press, 1976, pages 281.

[5] A. Huertas and R. Nevatia, "Edge Detection in Aerial Images Using $\nabla^2 G(x,y)$ Filters", Technical Report, USCIPI Report 1010, March, 1981.

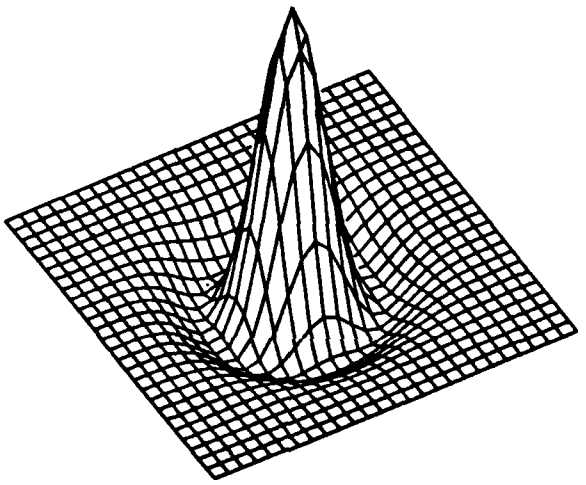


Figure 1: A Laplacian-Gaussian Filter with ω=7 and λ=27.

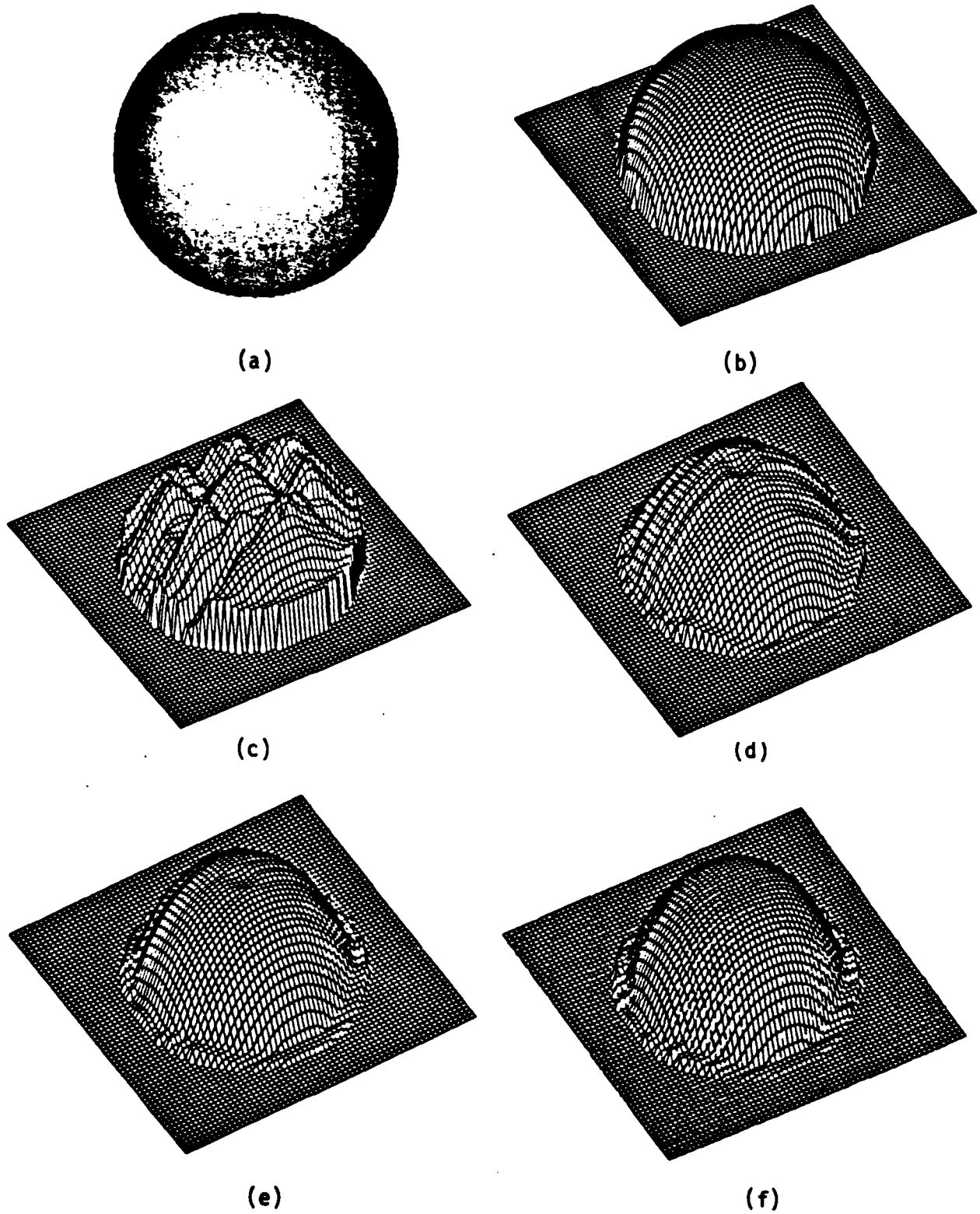
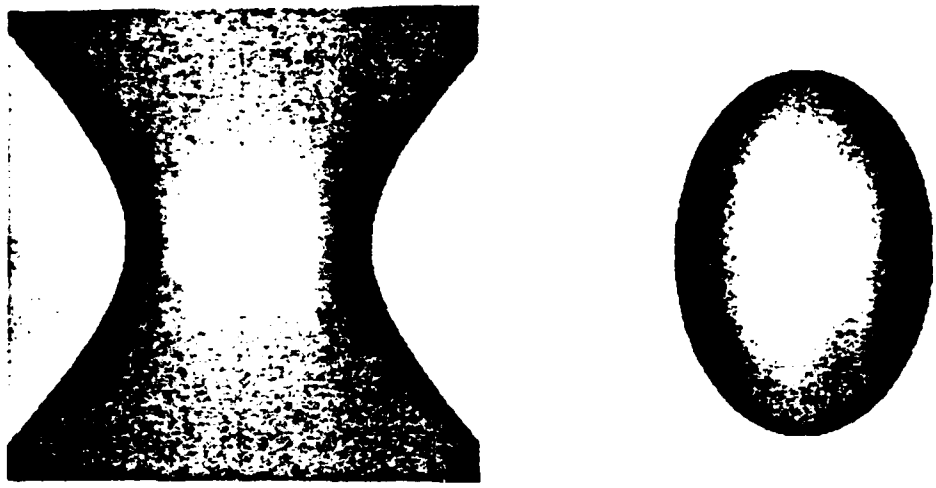
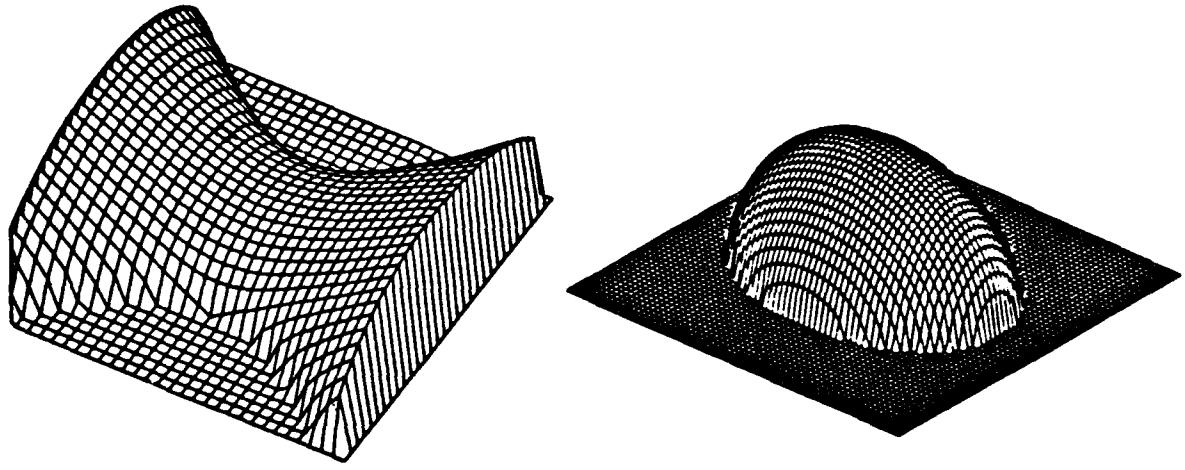


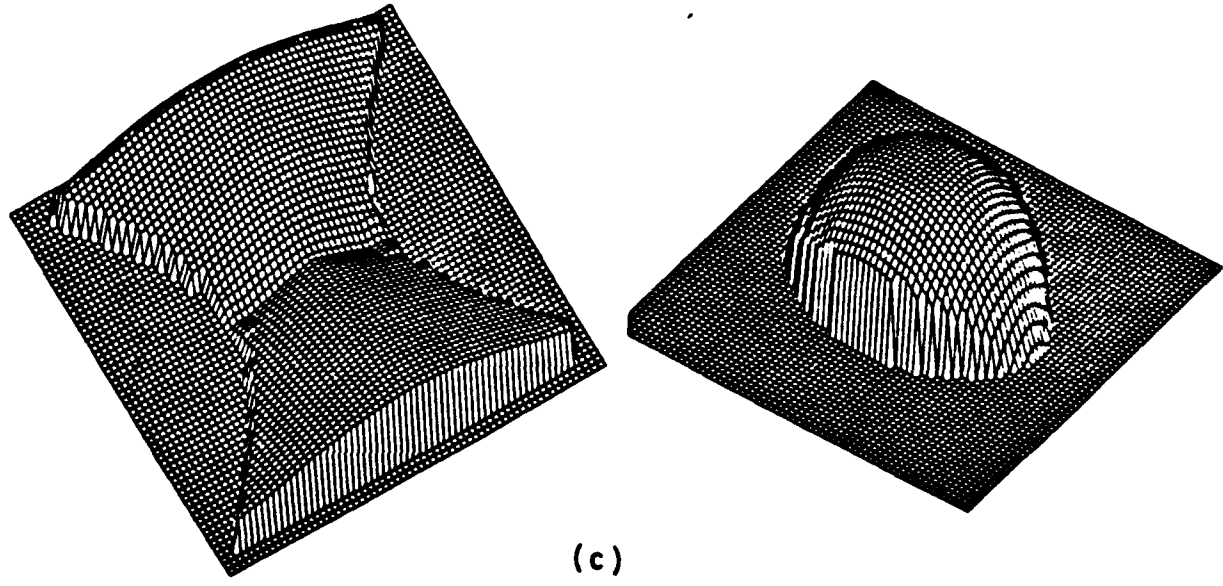
Figure 2: Recovered shapes from different size operators. The synthetic sphere in (a) with true surface in (b) is convolved with $\nabla^2 G$ having ω=1,3,3,5 and λ=3,5,7,15 pixels.(c)-(f) show the recovered shapes.



(a)



(b)



(c)

Figure 3: Shape recovered by local estimation for a hyperboloid and an ellipsoid. (a) shows the synthetic images,(b) the true depth maps,(c) the estimated shapes.

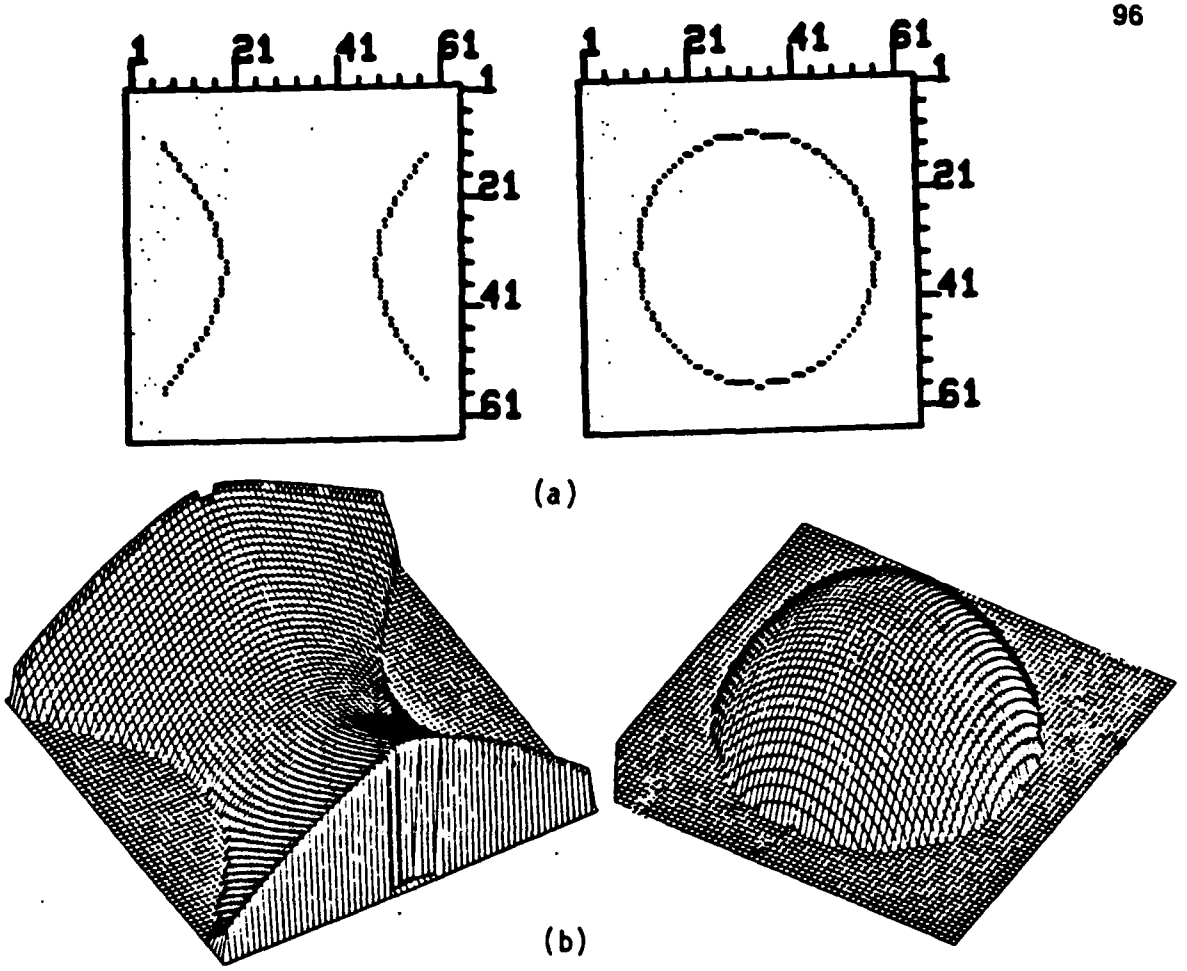


Figure 4: Shapes recovering for synthetic images of a sphere and a hyperboloid by global relaxation method. (a) Occluding boundaries,(b) Recovered shapes.

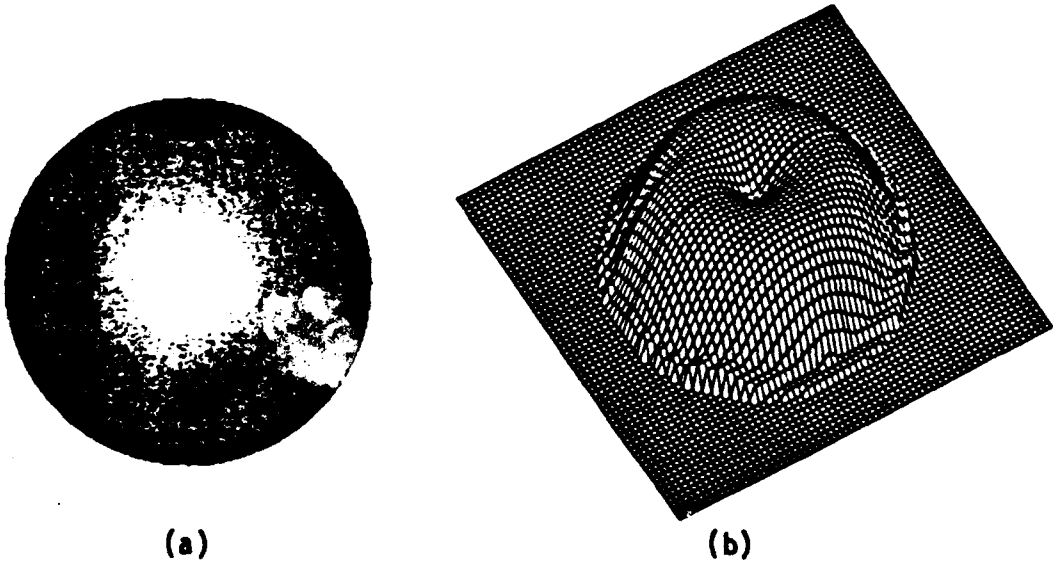


Figure 5: (a) A synthetic sphere with $I \propto 1-2i/\pi$, (b) recovered depth map by local analysis method.

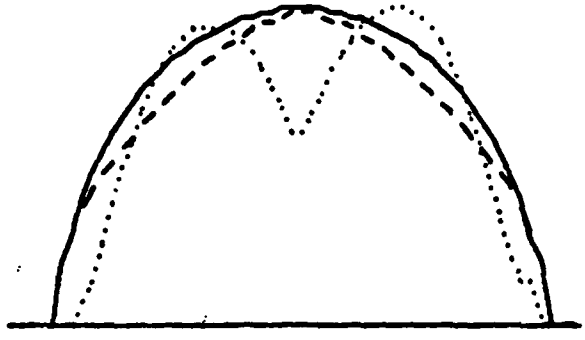
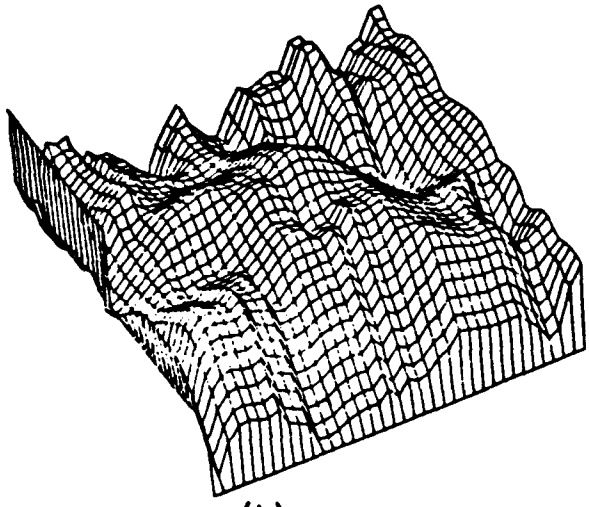
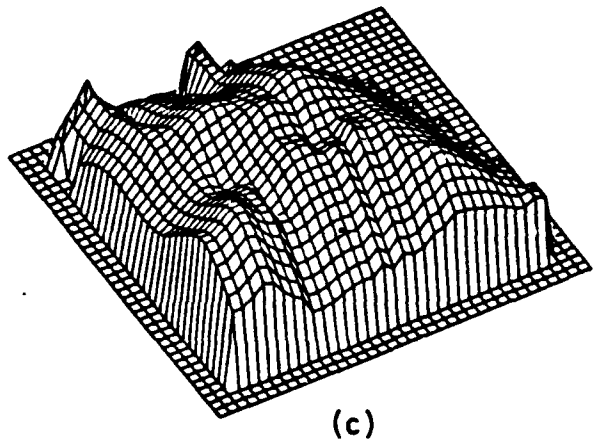


Figure 6: Profile of true shape and the shapes recovered from local estimation and global relaxation method. Solid line indicates true depth, dotted line indicates local analysis, and dashed line indicates global relaxation.

Figure 7: (a) Elaine - Shoulder, 76x76.

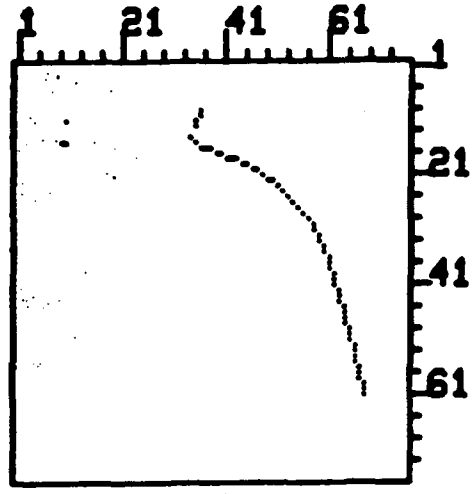


(b)

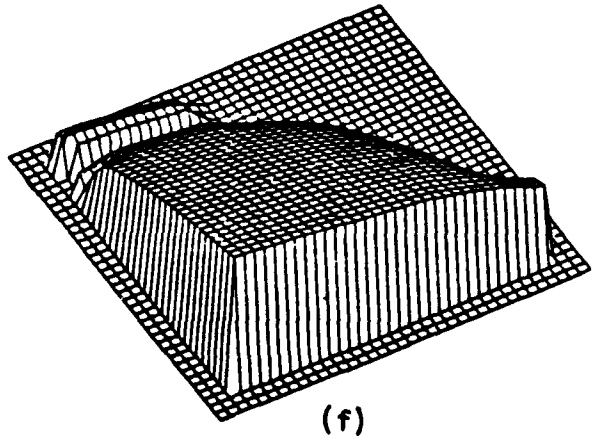


(c)

(d)



(e)

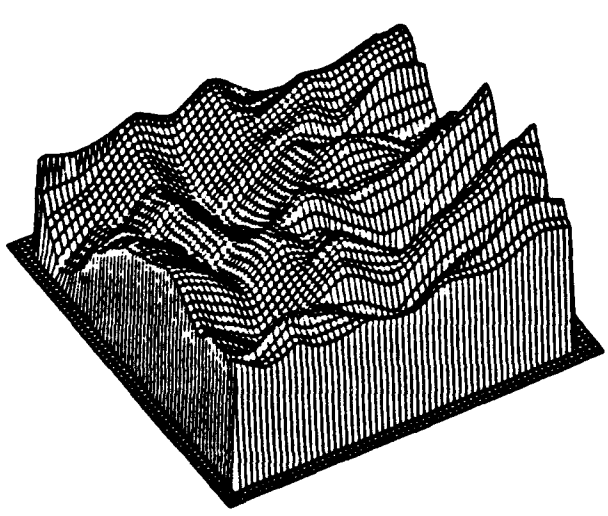


(f)

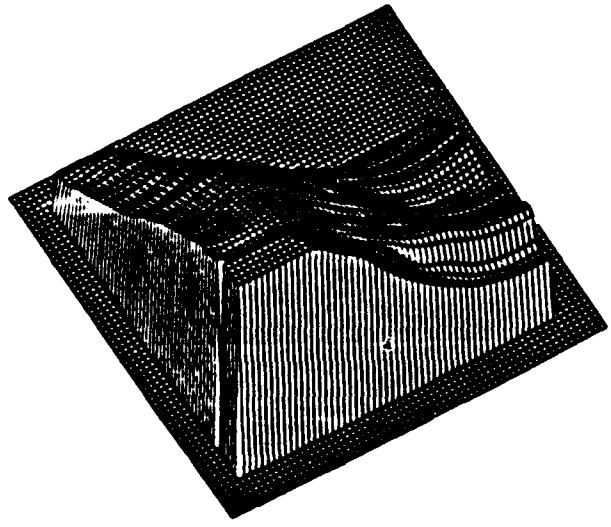
Figure 7: (continued)

(b) Recovered shape by local computation with background
(c) Recovered shape by local computation without background
(d) Segmented regions
(e) Occluding boundaries
(f) Recovered shape by global relaxation method

(a)



(b)



(c)

Figure 8: (a) Face ,64x64

(b) Recovered shape from local computation
(c) Recovered shape from global relaxation with initial values

SECTION 8

TEXTURED REGION EXTRACTION USING PYRAMIDS

Hong-Youl Lee

8.1 INTRODUCTION

Even though texture plays an important role in human visual perception, attempts to utilize texture in segmentation have been limited and have encountered many formidable problems. In many cases such problems stem from the very essential fact that textural properties at a location (or pixel) in the image are measured over some neighborhood of that location whereas intensity properties (e.g. spectral value, luminance, tristimulus values) are measured strictly on that location itself. For this reason, textural properties measured for a pixel near the boundary of two differently textured regions show mixed properties of the two regions.

The population of pixels with mixed properties often bring undesirable results when the conventional segmentation schemes which work well with the intensity properties are adopted to deal with texture. In attempts to use unsupervised pattern classification (clustering) techniques [1-3], for example, well separated and individually condensed clusters rarely exist in feature space. Neither do prominent peaks in property histograms when the histogram-based thresholding technique [4] is used. In the application of the split-and-merge algorithm [5-6], there is a danger of obtaining separated boundary regions if the merging criterion is set too strict or merging differently textured regions together if it is set too loose. Such is also true with the application of the pyramid-linking algorithm [7] where the number of segments to be obtained is controlled arbitrarily by setting the highest level of the pyramid differently.

Even though the fine details of the image are smeared by the texture processing, all these approaches were directed to divide the image into completely exhaustive uniform regions without specifically separating the textured portions from the untextured ones. One reasonable alternative can be extracting only the textured regions partially and conservatively without committing ourselves to the untextured portions which will be segmented more easily and accurately using intensity properties instead. Therefore we proposed a region extraction scheme which selects the most likely textured spots of the image as the starting elements and then expands them up to the boundaries to form separate connected regions on the previous report [8]. Several important changes were made ranging from the starting element selection to the texture measure itself.

8.2 PREDICTION OF TEXTURE PRESENCE AND UNIFORMITY TEXTURE MEASURES

Local uniformity and the change in uniformity at different resolutions are used as the texture cue in our extraction scheme. While constructing a level of the intensity pyramid where level L is obtained by non-overlapped block averaging of level L-1, the corresponding levels of the uniformity pyramid indicating the local uniformity at that

resolution and of the uniformity-change (UC) pyramid indicating the local uniformity change from the lowest level are computed as shown in Figure 1. The underlying supposition on which these measures are valid is that the averaging process in constructing a pyramid structure changes a large textured region into a uniform luminance region at the level where the averaging window approximately equals the collection of texture primitives in size. (The principle of halftone prints, which are highly textured dot patterns meant to be viewed from far enough to blur the texture, is essentially the same.) This is true only if the variations in the illumination and the primitive size are small throughout the region.

After subdividing the image into non-overlapping square windows, the average at each level of the uniformity pyramid taken over a window is used in determining the presence of large enough textured regions at the window location and estimating the proper level of resolution (level T) which is compatible with the size of the texture. If the average uniformity over a window improves at a higher level of the pyramid than the level 1 and the average UC over the window is greater than that over the entire image at the higher level, this window is marked as the probable site for a textured region. For each connected set of the textured windows, level T is determined by examining the average uniformity value over the entire set at each level. The size of the window is given by the human operator and depends on the expected size of the textured regions in the image. The low altitude aerial image of Figure 2, for example, has large forest regions which cover more than half the image and have approximately the same size of texture primitives. A very large window (e.g. 2x2 dividing) is desirable or even no dividing is necessary in this case where a single type of texture dominates the whole image. The starting elements shown in Figure 3 were actually obtained without dividing the image. The outdoor house image of Figure 7 has a large portion of untextured regions (sky, wall, car) and the textured regions have different size and texture primitive (trees at different distances, roofs, etc.). The starting elements shown in the figure were derived using 8x8 dividing.

8.3 EXTRACTION OF TEXTURED REGIONS

After the level T is determined for each connected set of the textured windows, three consecutive levels of the pyramids are involved in extracting compact textured regions. At level T+1, pixels with high uniformity values (e.g. upper 20% from the local histogram corresponding to the textured window set) are thresholded to form the starting elements. At level T, one of the starting elements (magnified by the factor of 2 to take account of the level descent) is grown by merging neighboring pixels whose uniformity and UC values lie inside the uniformity and UC ranges of the magnified element. Though it is unreliable to use the level T-1 uniformity and UC values inside the textured region, a textured region often adjoins untextured regions or regions with a texture of different primitive size, which are detectable at level T-1 (e.g. a forest region touching rivers or roads in an aerial image). At level T-1, therefore, boundary refining is carried out by eliminating the untextured or differently textured portions (with low uniformity or small UC values) from the search area which is constructed at level T and magnified by 2 to be compatible at level T-1. (After the region growing stops at level T, the search area is formed by the exterior boundary pixels as well as boundaries of holes and their neighboring pixels within a distance of 1.) After one region is extracted, the process is repeated

using another starting element which is separate from the detected regions. The starting elements from Figure 2 are shown in Figure 3 and the binary images of the resulting regions after each step are shown in Figure 4. Figure 5 shows the boundaries of the extracted regions on the original image. The final result on another test image (high altitude image of the San Francisco urban area) is shown in Figure 6. Figure 8 shows the final regions extracted from Figure 7.

8.4 CONCLUSIONS

The test results on natural images show that the proposed technique can extract large textured regions fairly well. Without the sophisticated description of textures from a stochastic or structural model, simple texture measures achieve sufficient results in the natural image domain.

REFERENCES

[1] S.G. Carlton and O.R. Mitchell, "Image Segmentation Using Texture and Gray Level," Proc. IEEE Conference on Pattern Recognition and Image Processing, June 1977, pp. 387-391.

[2] R.A. Jarvis, "Region Based Image Segmentation Using Shared Near Neighbor Clustering," Proc. IEEE SMC Conference, Sept. 1977, pp. 641-647.

[3] G.B. Coleman and H.C. Andrews, "Image Segmentation by Clustering," IEEE Proceedings, Vol. 67, May 1979, pp. 773-785.

[4] H.Y. Lee and K.E. Price, "Using Texture Edge Information in Aerial Image Segmentation," Technical Report USCISG 101, 1982.

[5] P.C. Chen and T.Pavlidis, "Segmentation by Texture Using a Co-Occurrence Matrix and a Split-and-Merge Algorithm," Computer Graphics and Image Processing, Vol. 10, 1979, pp. 172-182.

[6] P.C. Chen and T.Pavlidis, "Segmentation by Texture Using Correlation," IEEE PAMI Transactions, Vol. 5, 1983, pp. 64-69.

[7] M. Pietikainen and A. Rosenfeld, "Image Segmentation by Texture using Pyramid Node Linking," IEEE Transactions on Systems, Man, and Cybernetics, December 1981, pp. 822-825.

[8] H.Y. Lee, "Extraction of Textured Regions in Aerial Imagery," Technical Report USCISG 102, Oct. 1982.

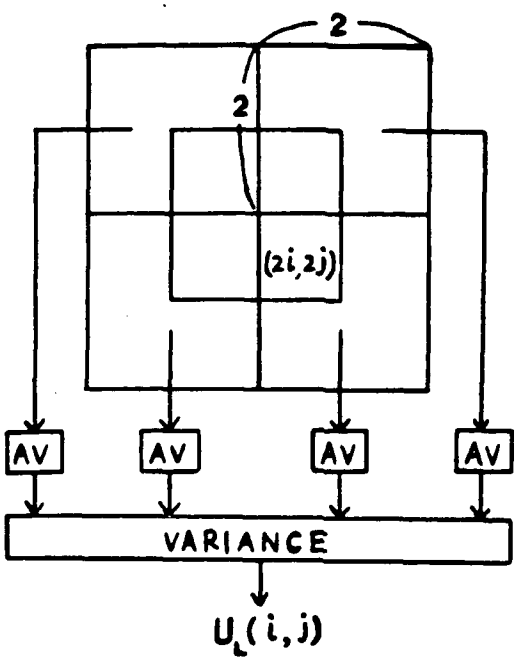


Figure 1: 4 by 4 block at level L-1 involved in the computation of level L features

A(k,ℓ) is the average inside the 2 by 2 block whose lower co-ordinate is (k,ℓ), i.e.,

$$A(k,\ell) = \tfrac{1}{4} \sum_{m=k-1}^{k} \sum_{n=\ell-1}^{\ell} G_{L-1}(m,n)$$

Level L intensity: $G_L(i,j) = a(2i,2j)$

Level L uniformity:

$$U_L(i,j) = \text{var}\{A(2i-1,2j-1), A(2i-1,2j+1), A(2i+1,2j-1), A(2i+1,2j+1)\}$$

Level L uniformity-change:

$$UC_L(i,j) = \begin{pmatrix}\text{var on the level 0 values inside the whole block}\end{pmatrix} - U_L(i,j)$$

Figure 2: Low altitude aerial image

Figure 3: Starting elements selected at level 4

Figure 4: Results from the largest starting element after each step

(a) starting element at level 4
(b) region grown from (a) at level 3
(c) search area of (b)
(d) after the elimination of the untextured portions at level 2

Figure 5: Region boundaries with small noisy regions removed and smoothing

Figure 6: Final result on a high altitude image of San Francisco area

Figure 7: Outdoor scene with its starting elements

Figure 8: Final region boundaries from Figure 7.

SECTION 9

MULTIPLE RESOLUTION IMAGE TEXTURE SEGMENTATION

Allan G. Weber and Alexander A. Sawchuk

9.1 INTRODUCTION

One of the basic requirements of any image understanding system is to be able to segment the image into regions that have some common characteristic. For many applications, the common characteristic is texture. While image texture segmentation has been studied extensively in the past, most of the research used data obtained by making measurements on the image at one particular resolution. The purpose of this research is to find a way to use data from multiple resolutions in hierarchical manner that increases the overall segmentation accuracy above that which is obtained at a single resolution.

9.2 CLASSIFICATION FEATURES

The segmentation of a textured image can be viewed as a constrained classification problem in which the constraints are derived from available spatial information. As with any classification problem, the proper selection of the features to be used in the classification algorithm is very important for obtaining the best possible results. To provide a solid basis for the rest of the research, we will use features similar to the texture energy measures developed by Laws [1]. His features have been shown to work as well or better than most others and are also relatively easy to calculate. The test image to be used is a texture mosaic (Figure 1) which consists of eight different textures: grass, water, sand, wool, pigskin, leather, raffia, and wood. All eight textures are present in the image in squares of size 128x128, 64x64, 32x32, and 16x16.

The texture energy features are based on a using a sequence of convolutional operators on the image data, one small (micro-window) and one large (macro-window). To calculate a classification feature, the image is first filtered (convolved) with the micro window operator. These are small convolution masks designed to act as matched filters for certain types of quasi-periodic variations commonly found in textured images. Typically these masks are of size 5 by 5 or smaller and are zero-sum, resulting in an filtered image which is zero mean. The masks are intended to be sensitive to visual structures such as edges, ripples, and spots. Because the micro-texture features are quasi-periodic, we expect strong variations about the mean output as a function of mask position for masks that are matched to the local texture. Thus the relevant information for texture discrimination is now present as the image variance. Therefore the second part of calculating the features involves measuring the local sample variance (or some approximation) within overlapping or non-overlapping macro-windows. It is the size of this macro-window that will be changed to give the multiple resolutions for the final classification. Typically, the macro-windows sizes will be on the order of 15 by 15 or 31 by 31.

The variance in the local windows of the filtered image can be measured in a variety of ways. The true sample variance within a 2n+1 by 2n+1 window at point (x,y) is given by

$$\sigma^2(x,y) = (2n+1)^{-2} \sum_{i=x-n}^{x+n} \sum_{j=y-n}^{y+n} (f(i,j) - m(x,y))^2. \quad (1)$$

where the mean, m, is given by

$$m(x,y) = (2n+1)^{-2} \sum_{i=x-n}^{x+n} \sum_{j=y-n}^{y+n} f(i,j). \quad (2)$$

Because the output of the small convolution masks is theoretically zero mean, the local variance may be approximated by assuming that the image is indeed zero mean and averaging the squares of the points within the window.

$$\sigma^2(x,y) = (2n+1)^{-2} \sum_{i=x-n}^{x+n} \sum_{j=y-n}^{y+n} (f(i,j))^2. \quad (3)$$

Experimental examination of the statistics of some filtered images show that this zero mean assumption is justified.

A final step in creating the feature set is a normalizing process. The normalizing factors are derived in much the same way as the feature points described above. As above, the original image is filtered with a small (5 by 5) smoothing convolution mask. However in this case the operator is not zero sum, resulting in a non-zero mean filtered image. The output of this operation is similar to that of a low-pass filter. The standard deviation, σ , of the output image is measured as described above except that the true variance (Equation 1) must be measured since the zero mean assumption is no longer valid. The resulting standard deviations values are used to normalize the feature images on a pixel by pixel basis. Since the final feature images are now a ratio of standard deviations, any difference in gain from one image to another will cancel. Bias offsets are also canceled due to the zero sum nature of the convolution masks.

In Laws' work, the texture energy features were calculated for several different micro-texture masks. In order to reduce the dimensionality of the classification process, a principal component transformation was then used to allow selection of the a working subset containing the most significant transformed features. This process has been simplified in our case. A subset of the micro-texture masks were selected beforehand and used without performing the principal component transformation. Figure 2 shows the micro window masks used. These correspond to the E5L5, R5R5, E5S5, and L5S5 masks used by Laws. Their performance is certain to be inferior to that which would be obtained using an equal number of features which are optimum linear combinations of many features. However for the purpose of this research, the relative performance of the classification at various resolutions is more important than the absolute classification accuracy. Figures 3 and 4 show the resulting features generated with the E5L5 mask for the texture mosaic image. These images have been scaled to an eight-bit range for the purpose of viewing. All the features used for the classification are stored as floating point numbers. The statistics for these features are given in Table 1.

Table 1: Normalized Feature Statistics for Texture Mosaic Image

Feature	Mean		Min	Max
E5L5 (31x31)	50.4	12.3	12.9	89.5
R5R5 (31x31)	20.8	7.8	6.8	75.2
E5S5 (31x31)	7.6	1.9	2.7	14.5
L5S5 (31x31)	36.5	9.9	17.1	96.4
E5L5 (15x15)	58.8	16.5	6.8	158.2
R5R5 (15x15)	21.8	10.7	4.7	110.0
E5S5 (15x15)	8.2	2.7	1.7	22.9
L5S5 (15x15)	39.7	14.4	10.3	156.0

9.3 CLASSIFICATION ALGORITHM

As with any pattern classification problem, the selection of the classification algorithm is based to a large extent on the type of information available for use. For most applications of texture classification one does not know beforehand what textures will be present in the image. However, for the initial attempts at multi-resolution classifying we will have available a priori the class statistics. This allows us to use a Bayes classifier. This classifier is based on Bayes decision theory and guarantees that no other decision rule will yield a smaller probability of error. In the following discussion, s is a discrete random variable which describes the classes present in the experiment and can take on values s_1, s_2, ... s_n. The probability density of s is given by P(s). The continuous random variable, x, represents the sample in question. The distribution of x is dependent on the class it belongs to. This gives the class-conditional probability density function, $p(x|s_i)$, which is the density of x given that x is a member of class s_i. According to Bayes decision rule, a point should be classified to class i if

$$P(s_i|x) > P(s_j|x) \quad \text{for all } j \neq i \tag{4}$$

or equivalently, by Bayes rule

$$p(x|s_i)P(s_i) > p(x|s_j)P(s_j) \quad \text{for all } j \neq i. \tag{5}$$

It not necessary to always compare the actual probability functions as shown above to make a decision. Instead we may define discriminant functions which will satisfy the same relationships but reduce the amount of calculations.

The classification of textures usually involves multi-dimensional data. The vector, **x**, contains the feature points from all features for the point in question. We will assume the features are Gaussian distributed with mean $\mathbf{m}_i$ and covariance matrices C_i. Under this assumption, it is convenient to define the discriminant functions as

$$g_i(x) = \log(p(x|w_i)P(w_i)) = \log(p(x|w_i)) + \log(P(w_i)) \tag{6}$$

$$= -\tfrac{1}{2}(x - m_i)^t C_i^{-1}(x - m_i) - \tfrac{d}{2}\log 2 \pi - \tfrac{1}{2}\log|C_i| + \log(P(w_i))$$

where d is the dimensionality of the feature set [2]

The $\frac{d}{2}\log 2$ term is common to all functions and can be removed. For the case where the classes are present in equal number, the a priori densities will be equal for all classes and the $\log(P(w_i))$ term can also be removed. This results in equivalent discriminant functions given by

$$g_i(x) = (x - m_i)^t C_i^{-1}(x - m_i) + \log|C_i|. \quad (7)$$

This function is basically Mahalanobis distance plus a class dependent bias. It is not a true distance metric since $g_i(m_i)$ is not equal to zero. The Bayes decision rule then classifies a point x to class i if

$$g_i(x) < g_j(x) \text{ for all } j \neq i. \quad (8)$$

The classifier described above was applied to the feature data calculated with 31 by 31 windows and with 15 by 15 windows. The classification results are shown in Figures 5 and 6. From these we can see that the classifier is performing about as expected. The classification using the 31 by 31 features is more accurate in the large regions but performs badly near the borders between the textures. The results in the small regions are essentially worthless. The classification with the 15 by 15 features is more accurate near the borders and in the small regions. However, there are also considerable errors in the interior of the large regions. Table 2 lists the accuracy of the classification for the overall image and for each sub-image containing regions of a particular size.

Table 2: Classification accuracy using Bayes classifier (%)

	31x31	15x15
Overall	68.13	69.69
128 x 128	83.29	78.21
64 x 64	69.26	71.07
32 x 32	48.41	58.22
16 x 16	23.75	44.31

A graph of the performance of the two sizes of features versus the region size would show an intersection at a region size slightly above 64x64.

9.4 CONFIDENCE MEASURE

The basic problem with a multi-resolution classifying scheme is that it is difficult to know when to use the large window classification result and when to use the small window classification result. From observing the result of classifying an image using an operator of a single size, one can see that the effectiveness of the operator is dependent on its position in the image. The larger operators are more accurate away from texture edges while the smaller operators perform better near texture edges. This leads to the simple rule: use the large operator result when classifying a pixel away from edges and use the small operator result for pixels near an edge. Unfortunately, since one of the reasons we are trying to classify the texture is to find the edges between them, we can never know in advance where the texture edges are. To get around this dilemma, the classification method is set up as a decision hierarchy in which certain classifications

have precedence over others. Specifically, the larger window operator result will always be used unless the estimated accuracy of the classification falls below a threshold. The key to making this type of decision process work is to be able to measure the confidence of the classification made with the large window operator. Ideally, this confidence measure will be high when the large window classification is correct and low when the classification is wrong.

The confidence measure is essentially a measure of "how close did this point come to matching a (hopefully) correct class?" It should also take into account the relative nearness of any other class means. For a minimum-distance-to-mean classifier using Euclidean or Mahalanobis distances, a confidence measure can be implemented using the discriminant function values (Equation 7). While the discriminant function is not a true distance metric, it serves much the same purpose and can be treated as one when calculating confidence values.

One possibility for a confidence measure would be

$$conf_A = 1 - \frac{\text{distance to nearest mean}}{\text{distance to second nearest mean}} \qquad (9)$$

This satisfies an intuitive criterion for a confidence measure in that for a point which falls exactly on a class mean the confidence is 1 and for a point midway between means the confidence is zero. Figure 7 shows the result of applying this confidence measure to the texture mosaic features generated with a 31 by 31 local standard deviation operator. In the image, low confidence is indicated by darker shades while high confidence shows up as brighter areas. It can be seen that this method tends to react very quickly to any texture edges. One possible deficiency is that the low confidence regions following the texture edges tend to be very narrow, in a sense overestimating the correctness of the decision. This confidence measurement also doesn't take into account the number of classes that came close to being selected. It will return the same confidence value for distances of 1, 1.2, 5, and 10 to the four closest means as it would for distances of 1, 1.2, 1.2, and 1.2.

Various modifications to this general confidence measure have been tried with marginally acceptable results. In Figure 8 we see the result of applying a confidence operator defined as

$$conf_B = 1 - \frac{\text{distance to nearest mean}}{\text{average distance to all 8 means}} \qquad (10)$$

to the same features. Using the average distance to all the means in the denominator does not yield very good results since the confidence values tends to be driven up by the presence of a class mean far away from the point in question. The confidence values in this example range from 0.99 down to only 0.43. Since we would not classify the point in question to the class whose mean is farthest from it, the confidence should not inadvertently benefit from its presence.

Figure 9 shows a compromise confidence measure calculated as

(11)

$$conf_C = 1 - \frac{\text{distance to nearest mean}}{\text{average distance to 3 nearest means}}$$

This method seems to come the closest to how one might imagine a confidence operator to act but still leaves something to be desired.

The most prominent problem areas that have shown up using this technique occur in the regions where the texture region is much smaller than the size of the operator window. The confidence calculation often results in all points in these areas getting a confidence level well above what they deserve. The cause for this seems to be due to the spatial arrangement of the class means in the feature space. In many cases, the mean of one class happens to be close to being on a line between two other class means. The effect of the mixture density of these two classes is usually to classify the point to the third in-between class. Since the confidence operation only sees that the point ended up relatively close to a class mean, the confidence is declared as being high. This problem is compounded further when there are several classes present in the operator window. Any class that happens to have its mean near the center of the means of the classes in the the window tends to collect most of the class assignments. An example of this can be seen in Figure 10 where the means in the feature space have been projected onto the E5L5-R5R5 plane for plotting. We can see that the class means for textures 1 and 3 through 7 are well separated from those for textures 2 and 8 with the texture 2 mean lying roughly between that of texture 8 and the others. With this spatial arrangement, many points in texture 8 that lie near a texture edge will get classified using a mixture of texture 8 statistics and statistics of one or more of the other classes. The feature point ends up lying somewhere between the mean of class 8 and the means of the neighboring texture. As can be seen from the graph, a point that lies between the texture 8 mean and the means of textures 1 and 3 through 7 will probably be closer to the mean of texture 2 then it will be to that of texture 8. For this reason, the points near the border of a texture 8 region are usually classified incorrectly as belonging to texture 2.

One piece of information that is perhaps not properly considered with this confidence operator is the number of classes that have means close enough to the mean of the selected class to have been under consideration. The measurement defined by equation 11 will return the same confidence level for distances of 1, 1.1, and 6.9 to the three closest means as it will for distances of 1, 4, and 4. Intuitively, we should feel less confident about the decision made in the first case than we would about the second.

9.5 SPATIAL COHERENCE

When segmenting a textured image, it is a safe (and probably necessary) assumption that the textures will be present in cohesive regions of some minimum size as opposed to scattered and intermixed pixels. The classification techniques we have used so far are all based on individual pixels. The classification decision for one pixel is not dependent on the decision made for any neighboring pixel nor is any consideration made of the spatial arrangement of the data. This is obviously different from the way that a human observer would classify textures. A human would probably rely greatly on the classification of the area surrounding the pixel in order to make the best possible decision. Since we assume the textures are in regions, this is certainly a good idea and one that should be implemented in any texture classification technique. If a point is believed to belong to class 1 but all the neighboring pixels in a surrounding region have been classified to class 2, then we should strongly consider classifying the pixel to class 2. No matter how we go about taking the surrounding neighborhood into consideration, it will probably be possible to invent a situation where we would have been better off ignoring the neighboring pixels, but in general this information can be put to use in lowering the rate of classification errors.

Work is currently in progress to implement these ideas. The basic concept involves measuring the "coherence" between a pixel and its neighbors. Since we are working with a multi-class problem, the measurement must examine the coherence between a pixel and its neighbors for each possible class assignment for the pixel in question. The resulting vector of coherence values represents a "coherence histogram" in which the value in each histogram bin indicates the coherence of the center pixel with the pixels around it under a different assumption for its class assignment. Due to the nature of the coherence histogram in that it contains information about the spatial distribution of the texture classes, it will be a major component in the final decision process. We will describe work on using spatial coherence in a future report.

9.6 DECISION PROCESS

The final step in the segmentation process is to combine the previously developed information to select a final classification for the texture points. These sources of information include:

- The classification choice using the large window operators.
- The confidence of the large window operator classification.
- The coherence histogram giving spatial information about the surrounding area.
- The classification features from the small window operators.

The basic decision process can be stated as follows: If the confidence of the large window operator classification is above a threshold use that classification, otherwise reclassify using the small window operators with the allowable class assignments determined by the spatial information from the coherence histogram. Implementing this process basically involves the selection of the threshold and the design of the class elimination rule. Work is currently in progress to develop an acceptable decision rule which uses all the information listed above.

Even without the use a complex decision rule, the desirability of using multiple resolution features can be demonstrated by using the results of the 31 by 31 and 15 by 15 window classifications and the confidence values. A simple multiple resolution classifier can be implemented by using the confidence value as a threshold for selecting a classification from one of the two images. If the confidence value is above the threshold then the large window classification result will be used. If the confidence value is below the threshold then the small window classification result will be used. The confidence value in this example is given by Equation 11. The results of using this very simple rule with a threshold of 0.6 are shown in Figure 11 and Table 3. Comparison with the classification results using a single size operator (Table 2) shows that this does provide an improvement in all areas of the image, except in the region of highest resolution (16x16).

Table 3: Classification accuracy using Bayes classifier (%)
Combining single resolution results using confidence threshold

Overall	73.75
128 x 128	85.30
64 x 64	74.04
32 x 32	59.22
16 x 16	41.48

9.7 SUMMARY AND CONCLUSIONS

The multiple resolution segmentation process is based on the fact that features from different resolutions will behave differently in many areas of an image. The basic idea is to always be using the features that can be expected to be most accurate in a particular area of the image. This goal is accomplished by using a confidence operator to estimate the accuracy of the classification at one resolution level. If the confidence is not high enough, then the classification is performed using the features from the other resolution. A spatial coherence measurement is used to constrain the final classification to be one that is consistent with the surrounding area.

The work that has been done so far has provided a solid basis for investigating the effects of different coherence measurements and decision rules. Several different coherence measuring techniques are being tried and some seem to be exhibiting the required characteristics. Preliminary attempts at using the coherence histogram in a decision process have been encouraging, and will be described in detail in future reports.

REFERENCES

[1] K.I. Laws,"Textured Image Segmentation," PhD Thesis, University of Southern California, 1980.

[2] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis, Wiley-Interscience, 1973.

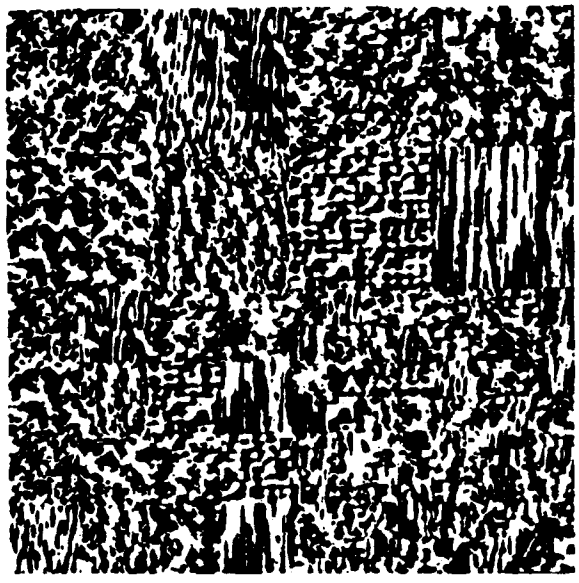


Figure 1: Texture mosaic image

-1	-4	-6	-4	-1
-2	-8	-12	-8	-2
0	0	0	0	0
2	8	12	8	2
1	4	6	4	1

E5L5

1	-4	6	-4	1
-4	16	-24	16	-4
6	-24	36	-24	6
-4	16	-24	16	-4
1	-4	6	-4	1

R5R5

1	0	-2	0	1
2	0	-4	0	2
0	0	0	0	0
-2	0	4	0	-2
-1	0	2	0	-1

E5S5

-1	0	2	0	-1
-4	0	8	0	-4
-6	0	12	0	-6
-4	0	8	0	-4
-1	0	2	0	-1

L5S5

Figure 2: Micro window masks

Figure 3: 31 by 31 E5L5 normalized feature

Figure 4: 15 by 15 E5L5 normalized feature

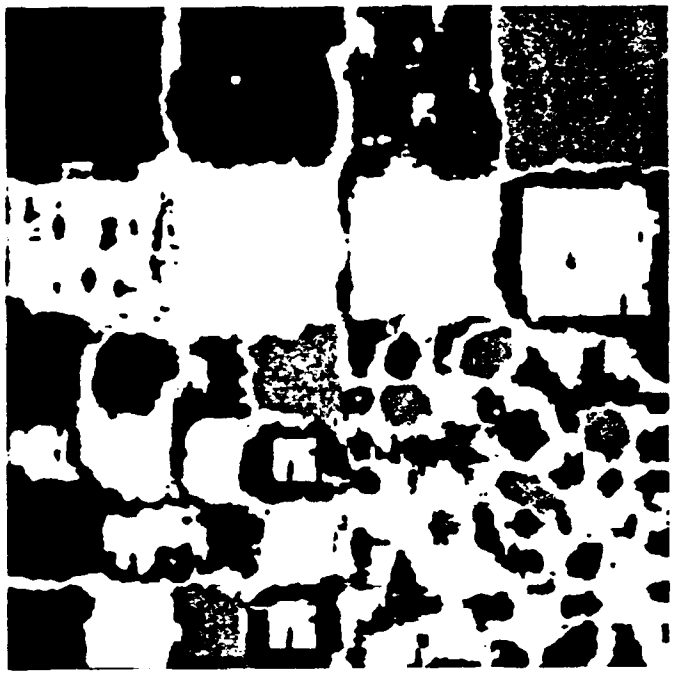


Figure 5: 31 by 31 classification results

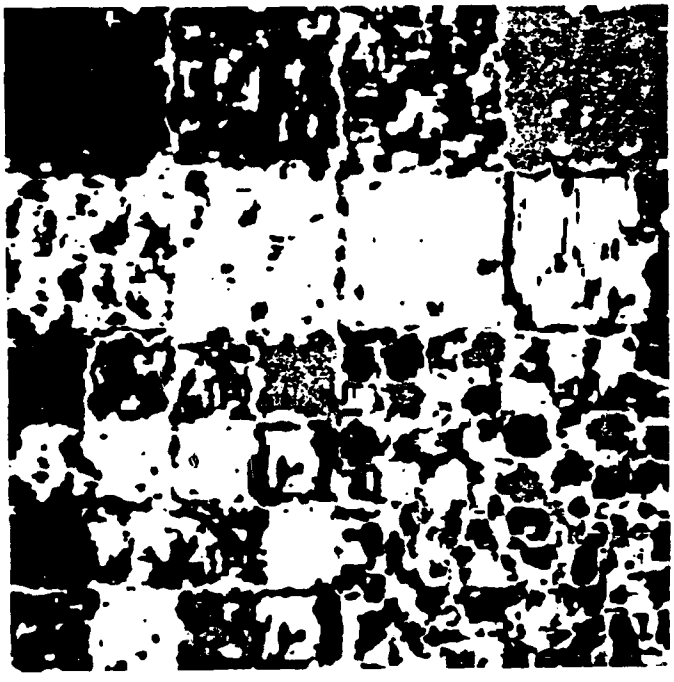


Figure 6: 15 by 15 classification results

Figure 7: $Conf_A$ applied to 31 by 31 features

Figure 8: $Conf_B$ applied to 31 by 31 features

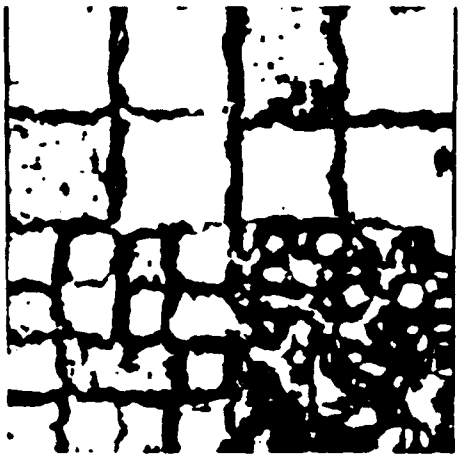


Figure 9: $Conf_C$ applied to 31 by 31 features

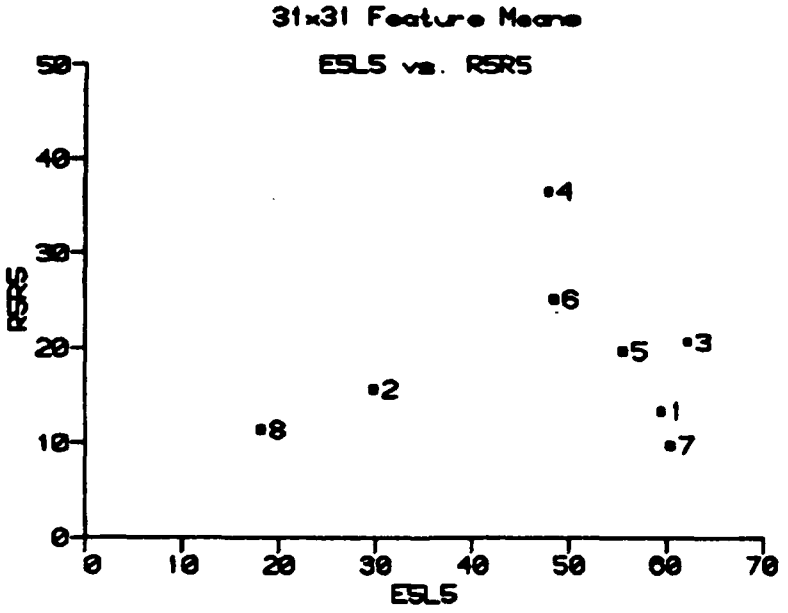


Figure 10: Projection of means onto E5L5-R5R5 plane

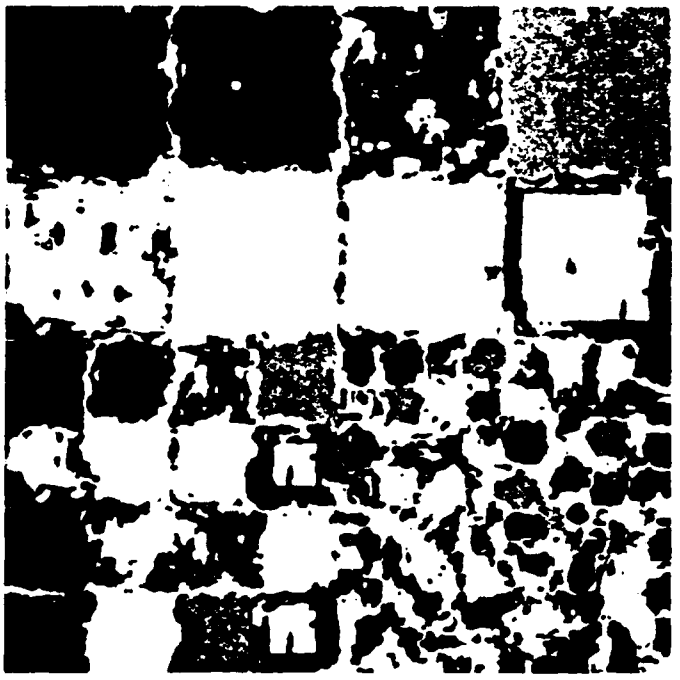


Figure 11: Hierarchical classification based on confidence threshold

SECTION 10

A VLSI ALGORITHM AND ARCHITECTURE FOR SUBGRAPH ISOMORPHISM

Dan I. Moldovan

10.1 INTRODUCTION

10.1.1 Statement of the Problem

Consider two graphs $G_a = (V_a, E_a)$ and $G_b = (V_b, E_b)$ where V denotes the set of verteces (or points) and E denotes the set of edges (or lines) for each graph. The adjacency matrices for the two graphs are respectively $A = [a_{ij}]$ and $B = [b_{ij}]$. Two graphs G_a and G_b are said to be isomorphic to each other if there exists a 1:1 correspondence between the points of G_a and G_b that preserves adjacency. This definition implies that the two graphs have the same number of points and lines. If the number of points in the two graphs is not the same, then it may be possible that the smaller graph is isomorphic to a part of the larger graph. This is the problem of subgraph isomorphism; it is more general and includes the graph isomorphism as a particular case. In this paper we will assume that G_a is the smaller graph, i.e., $n_a \leq n_b$, where n is the number of nodes.

10.1.2 Complexity Considerations

Many possible algorithms for subgraph isomorphism can be deviced. One approach is to verify if adjacency matrix A can be made identical to a submatrix of the larger adjacency matrix B by permuting the rows and columns of B. The complexity of this operation is $0(n_b!)$. Algorithms with such exponential complexity are inefficient. An acceptable algorithm for subgraph isomorphism should be $0(n_b^k)$.

Many researchers have proposed algorithms for graph isomorphism problem. Two annotated bibliographies on this subject by Read and Corneil [1] and Gati [2] discuss more than 60 representative papers. Graph isomorphism is known to be equivalent to many other graph related problems, for instance automorphism partitioning problem [3].

10.1.3 Applications of Graph Isomorphisms

Since graphs can be used to represent large classes of structures, the isomorphism problem has many applications. Thus, graph isomorphism is useful in computer vision analysis, pattern recognition, artificial intelligence, robotics, chemistry and many other fields. This paper was particularly motivated by the necessity to compare images, that is, to determine if an image or scene taken by a camera corresponds to another expected scene. In computer vision this is called matching problem, and is employed in systems such as Acronym [4] and others. For large images, this problem is computationally intensive, and oftenly constitutes an impediment for the real-time realization of some computer vision algorithms.

In this paper we propose a parallel algorithm and architecture for the problem of subgraph isomorphism. The algorithm is relatively simple and requires simple logic. The algorithm has been designed having in mind requirements imposed by the VLSI technology, such as logic regularity, reduced and localized data communications and reduced input/output transfers. Indeed, the architecture proposed in this paper is easily realizable in VLSI.

10.2 PARALLEL ALGORITHM

10.2.1 Algorithm Development

The algorithm for subgraph isomorphism is based on manipulations of adjacency matrices. The permutations of rows and columns of one adjacency matrix is equivalent to a tree search. Our approach is to improve upon such brute force method by testing a necessary condition for isomorphism.

Define an $n_a \times n_b$ permutation matrix $M^* = [m_{ij}]$ whose elements are 1's and 0's. This boolean matrix is such that each row contains one 1 and no column contains more than one 1. Define a new matrix $C = [c_{ij}]$ obtained by permuting the rows and columns of B.

$$C = M^* B M^{*T}$$

G_a is isomorphic to a subgraph of G_b iff A = C. Thus, the isomorphism problem is to identify all possible permutation matrices M^* that will lead to A = C. The brute force solution to this problem is to perform a tree search on an initial matrix $M^0 = [m_{ij}]$ which contains all existing isomorphisms, if any. Such M^0 can be constructed as follows.

$$m^0_{ij} = \begin{cases} 1 & \text{if the degree of the jth point in } G_b \text{ is greater than or equal to the degree of the } i^{th} \text{ point in } G_a \\ 0 & \text{otherwise} \end{cases}$$

Ullmann [5] proposed a refinement procedure which significantly reduces the tree search method. This refinement procedure tests the following necessary condition: if node i in graph G_a corresponds through an isomorphism to node j in graph G_b, then all nodes in G_a adjacent to i must have correspondent nodes in graph G_b which are adjacent to j. This means that for any $m_{ij} = 1$ in matrix M for which this condition is not satisfied $m_{ij} = 1$ is changed to $m_{ij} = 0$. This simple idea can be translated into the following boolean matrix equations.

$$R = M_i \times B$$

$$S = A \times \bar{R}$$

$$M_{i+1} = M_i \bullet \bar{S}_i$$

where x indicates boolean product and • indicates logic AND. These equations contain a high degree of parallelism because it is possible to test the necessary condition at all nodes simultaneously. The algorithm uses equation (1) in an iterative manner; M_1 is simplified to M_{i+1} during iteration i.

Notice that M_{i+1} does not have more 1's than M_i, it can only have less 1's or be identical to M_i.

As a result of the refinement procedure only one of the following possibilities exist.

a) M_{i+1} has less 1's than M_i, but M_{i+1} is not an isomorphism yet.

b) $M_{i+1} = M_i$, i.e. no change made.

c) M_{i+1} has a row with 0's or two or more identical rows with only one 1. These conditions indicate that there is no isomorphism obtained from M_{i+1}.

d) M_{i+1} is an isomorphism.

For each of these possibilities some actions are taken by the algorithm. The actions taken can best be described by the following primitives.

Actions:

1. Repeat refinement procedure.

$$M_i \leftarrow M_{i+1}$$

2. Split M_{i+1} into M'_{i+1} and M''_{i+1} such that M'_{i+1} has one more row with only one 1, than $M_{i+1} \bullet M_{i+1} = M'_{i+1} + M''_{i+1}$ (+ is logic OR)

$$M_{i+1} \leftarrow M'_{i+1}$$

$$\text{LIFO} \leftarrow M''_{i+1}$$

3. Fetch a new matrix from LIFO

$$M_{i+1} \leftarrow \text{LIFO}$$

4. Send M_{i+1} to ouput because it is an isomorphism

$$\text{OUTPUT} \leftarrow M_{i+1}$$

The algorithm is illustrated in the flowchart shown below. It consists of a sequence of computations, tests and actions.

The computations refer to the refinement procedure, i.e. equations (1). These equations are highly parallel. The tests are as follows:

T_1: Test M_{i+1} if at least one row has 0's.

T_2: Test if there are two identical rows having only one 1 and rest 0's.

T_3: Test if M_{i+1} is an isomorphism.

T_4: Test if $M_{i+1} = M_i$

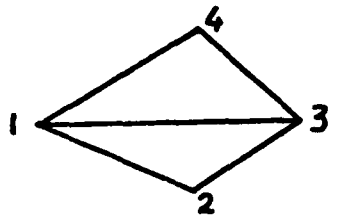
Based on these tests, some actions are taken as shown in the flowchart. As a termination procedure for this algorithm we can test if the LIFO is empty or not. If LIFO is empty it means that all possibilities have been exhausted.

A program was written to verify the efficiency of this algorithm. Thus, several subgraph isomorphism problems were studied and satisfactory results were recorded. Here are a few examples

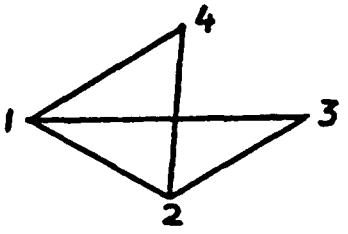
Example 1

Consider two graphs G_a and G_b with their adjacency matrices A and B respectively.

G_a:



G_b:



$$A = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 1 & 0 & 1 & 0 \end{array} \qquad B = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 0 \\ 4 & 1 & 1 & 0 & 0 \end{array}$$

For these graphs, an initial matrix M^0 can be constructed as indicated above.

$$M^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 1 & 1 & 1 \\ 3 & 1 & 1 & 0 & 0 \\ 4 & 1 & 1 & 1 & 1 \end{array}$$

In figure 2 it is shown the search tree which normally results for two 4th order graphs. In our algorithm, the refinement procedure systematically eliminates those branches which do not lead to any isomorphism. The algorithm first splits M^0 into M_1 and M_2

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad M_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

M_2 is stored in LIFO, and refinament procedure is applied to M_1. M_1 is unchanged after refinament procedure, and M_1 is splitted into M_{12} and M_{12}

$$M_{12} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \qquad M_{12} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Refinement procedure is applied to M_{12} while M_{12} is stored. It is found that M_{12} does not lead to any isomorphism and then M_{12} is fetched from LIFO. The process continues and eventually it is found that four isomorphisms exist for these two graphs. The paths from M^0 to these isomorphisms are marked with heavy lines in figure 2.

Isomorphism found:

G_a	G_b	G_b	G_b	G_b
1	1	1	2	2
2	3	4	1	3
3	2	2	3	1
4	4	3	4	4

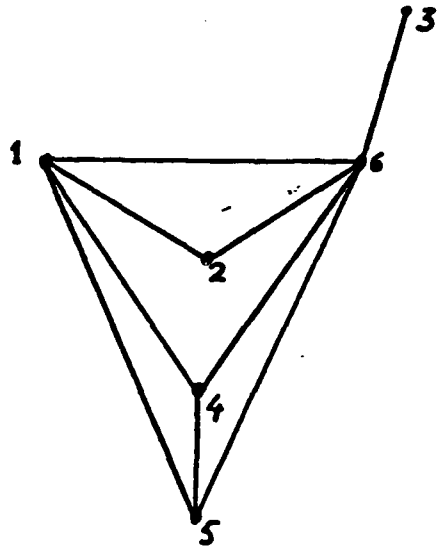
Number of pushes/pops 5

Number of refinments 17

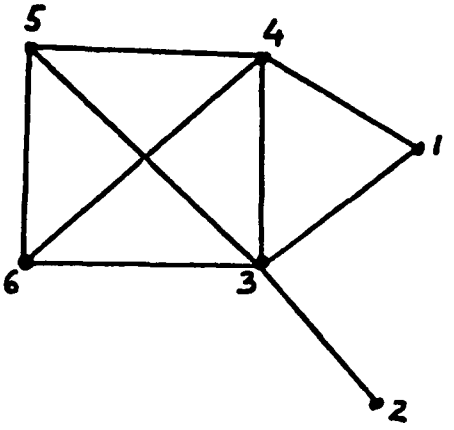
Example 2

Consider the graphs:

G_a:



G_b:



$$A = \begin{bmatrix} 0&1&0&1&1&1 \\ 1&0&0&0&0&1 \\ 0&0&0&0&0&1 \\ 1&0&0&0&1&1 \\ 1&0&0&1&0&1 \\ 1&1&1&1&1&0 \end{bmatrix} \quad B = \begin{bmatrix} 0&0&1&1&0&1 \\ 0&0&1&0&0&0 \\ 1&1&0&1&1&1 \\ 1&0&1&0&1&1 \\ 0&0&1&1&0&1 \\ 0&0&1&1&1&1 \end{bmatrix} \quad M^0 = \begin{bmatrix} 0&0&1&1&0&0 \\ 0&0&1&0&0&0 \\ 1&1&1&1&1&1 \\ 0&0&1&1&1&1 \\ 0&0&1&1&1&1 \\ 0&0&1&0&0&0 \end{bmatrix}$$

Isomorphisms found:

G_a	G_b	G_b
1	4	4
2	1	1
3	2	2
4	5	6
5	6	5
6	3	3

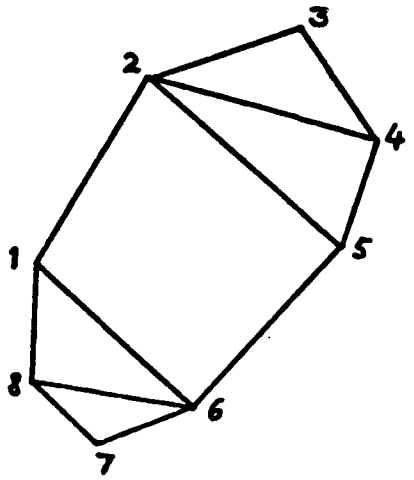
Number of pushes/pops 17

Number of refinements 66

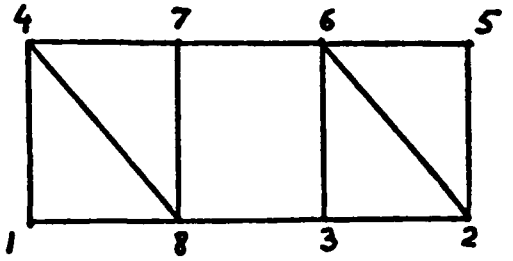
Example 3

Consider the graphs:

G_a:



G_b:



$$A=\begin{bmatrix}0&1&0&0&0&1&0&1\\1&0&1&1&1&0&0&0\\0&1&0&1&0&0&0&0\\0&1&1&0&1&0&0&0\\0&1&0&1&0&1&0&0\\0&0&0&0&0&1&0&1\\1&1&1&1&1&1&1&1\\0&1&1&1&0&1&1&1\end{bmatrix} \quad B=\begin{bmatrix}0&0&0&1&0&0&0&1&0\\0&0&1&0&1&1&0&0&0\\0&1&0&0&0&1&0&1&1\\1&0&0&0&0&0&1&1&0\\0&1&0&0&0&1&0&0&0\\0&1&1&0&1&0&1&0&0\\0&0&0&1&0&1&0&1&1\\1&0&1&1&0&0&1&0&0\end{bmatrix} \quad M^0=\begin{bmatrix}01&1&1&0&1&1&1\\00&0&0&0&1&0&1\\11&1&1&1&1&1&1\\01&1&1&0&1&1&1\\01&1&1&0&1&1&1\\00&0&0&0&1&0&1\\11&1&1&1&1&1&1\\01&1&1&0&1&1&1\end{bmatrix}$$

Isomorphisms found:

G_a	G_b	G_b	G_b
1	3	1	7
2	8	2	6
3	1	3	5
4	4	4	2
5	7	5	3
6	6	6	8
7	5	7	1
8	2	8	4

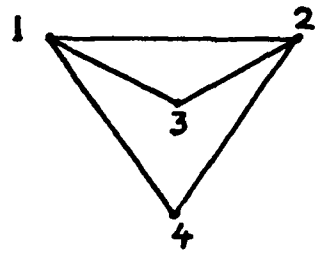
Number of pushes/pops 6

Number of refinements 54

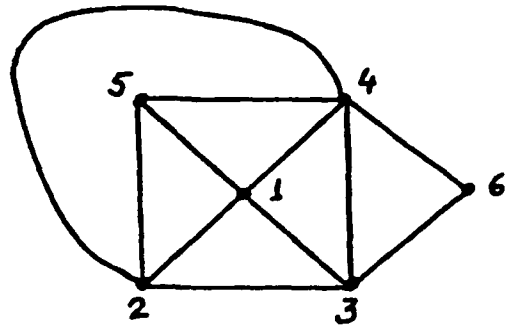
Example 4

Consider the graphs:

G_a:



G_b:



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Number of subisomorphisms found:	68
Number of pushes/pops	93
Number of refinements	228

End of examples.

It is difficult to measure exactly the efficiency of this algorithm because of the variable number of computations for different graph pairs. However, in general, we would like that the algorithm uses the stack memory as little as possible, such that the ratio between the number of occurrences of refinement procedure and stack operations is large. This is because for every iteration the refinement procedure is likely to eliminate unsuccessful paths. Notice from our examples that in this sense the efficiency of the algorithm increases with the number of nodes.

10.3 VLSI ARCHITECTURE

The block diagram of the architecture proposed to implement the algorithm described above is shown in figure 3. This architecture consists of:

- systolic-type array for computing equations (1)
- test and control logic for performing tests T_1 thru T_4 and for providing control sig-

nals accordingly
- LIFO stack memory for temporary storage
- switch for control of data communication.

All these main blocks contain a large degree of parallelism and regularity. This is the reason why this architecture is suitable for VLSI implementation. In what follows we will focus mostly on the systolic array which is used to implement equations (1). The rest of the blocks are relatively straightforward. A methodology for designing algorithmically specialized processor arrays has been described recently by Moldovan [6]. According with this technique, an algorithm containing loops can be mapped into hardware via some transformations of algorithm data dependences. Applying such techniques to equations (1) we arrived at the array shown in figure 4.

In this array matrix B moves south and matrix M_i moves east. Their boolean product is performed in a similar way as any matrix product, except that the cells are much simpler here, they perform only logic operations on bits. The resulting boolean product R moves south and it is feedback as shown in figure 4. Next, the boolean product S = A x R is performed, R moves south and A moves east. The operation $M_{i+1} = M_i$ S is performed either inside of the array or outside as part of the test and control procedure. The tests T_1 thru T_4 are easily implemented with combinational logic. The LIFO stack consists of two-directional shift registers. The capacity of LIFO is $O(n^3)$ bits where n is the size of the problem (number of nodes in larger graph.)

The switch placed at the output of test and control logic is used to route the newly computed M_{i+1} either to stack, back to systolic array or outside if an isomorphism occurs.

10.4 CONCLUSIONS

Subgraph isomorphism is in general considered to be a time consuming computational problem. The algorithm introduced in this paper performs a tree search while testing a simple necessary condition for isomorphism. Fortunately, most of the computations in each iteration can be performed in parallel due to the lack of data dependences between operations. A VLSI architecture consisting of a systolic-type array, stack memory and control logic was briefly discussed. This architecture can be integrated on a simple chip for practically large order graphs. Currently, we are investigating the design details. One pending issue is how to accommodate efficiently graphs with variable number of nodes. The algorithm described in this paper can be extended to clique detection, directed graph isomorphisms and perhaps other graph algorithms.

REFERENCES

[1] Read, R.C. and Corneil, D.G., "The Graph Isomorphism Disease," Journal of Graph Theory, Vol. 1., 1977, pp. 339-369

[2] Gati, G., "Further Annotated Bibliography on the Isomorphism Disease" Journal of Graph Theory, Vol. 3, 1979, pp. 95-109

[3] Karp, R.M., "On the Computational Complexity of Combinational Problems," Networks 5, 1975, pp. 45-68.

[4] Brooks, R.A., "Symbolic Reasoning Among 3-D Models and 2-D Images," Ph.D. Thesis, Stanford Univ., June 1981.

[5] Ullman, J.R., "An Algorithm for Subgraph Isomorphism," JACM, Vol. 23, No. 1, January 1976, pp. 31-42.

[6] Moldovan, D.I., "On the Design of Algorithms for VLSI Systolic Arrays," Proceedings of the IEEE, Vol. 71, No. 1, January 1983, pp. 113-120,(special issue on VLSI).

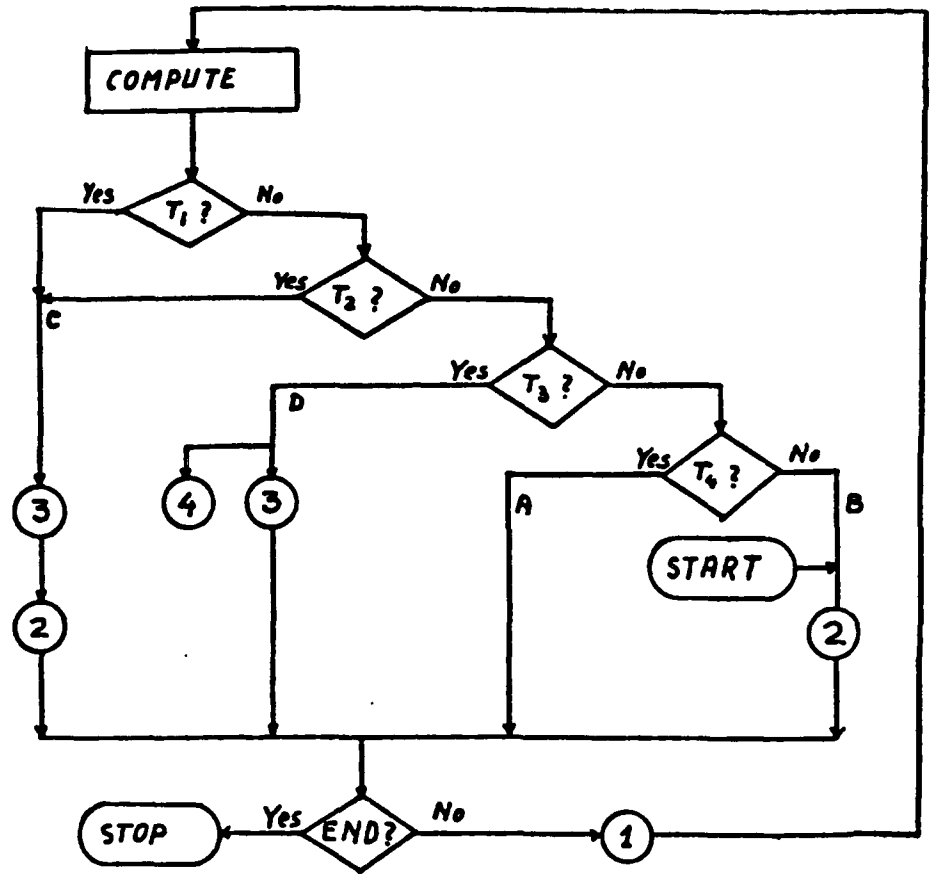


Figure 1: Algorithm for subgraph isomorphism

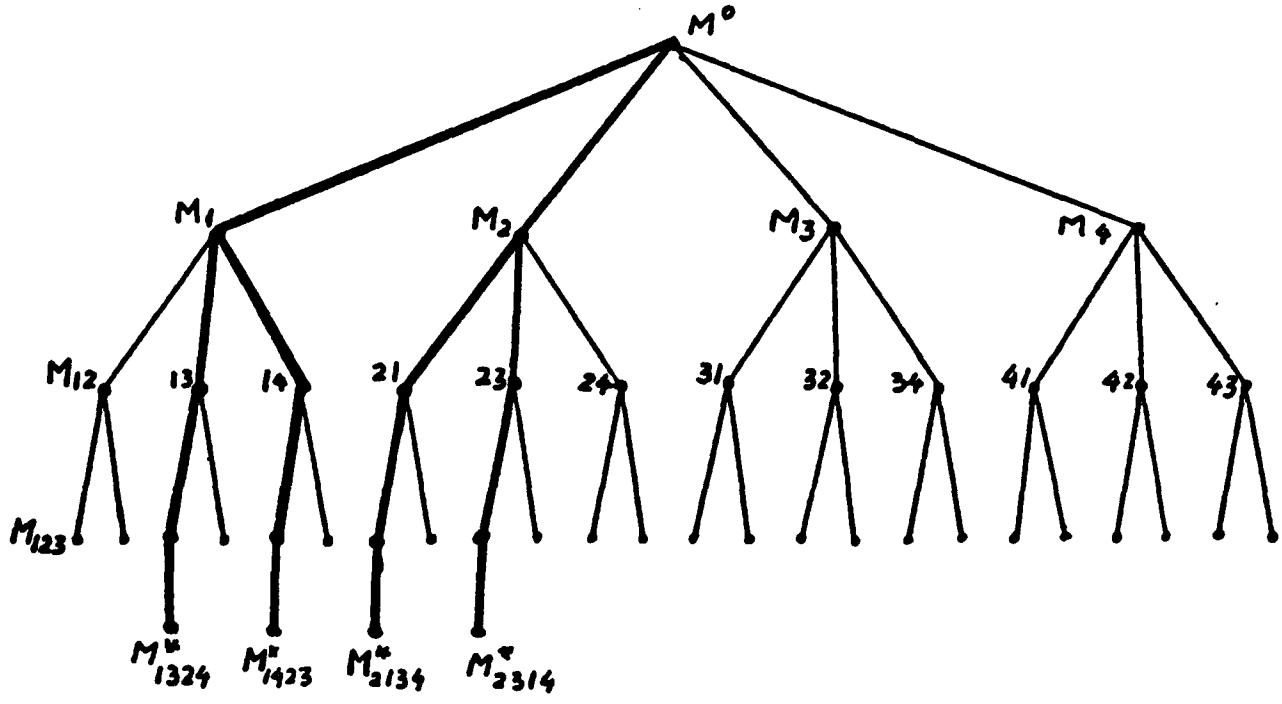


Figure 2: Tree search for example 1

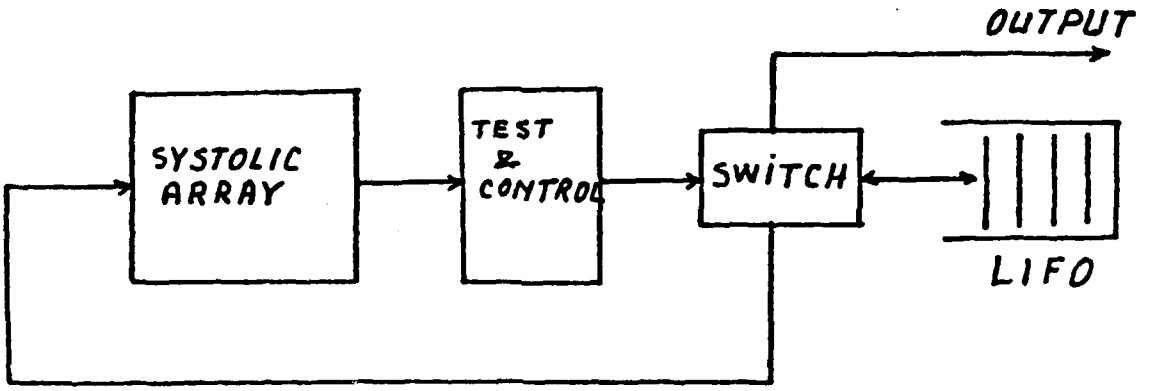


Figure 3: Architecture for subgraph isomorphism

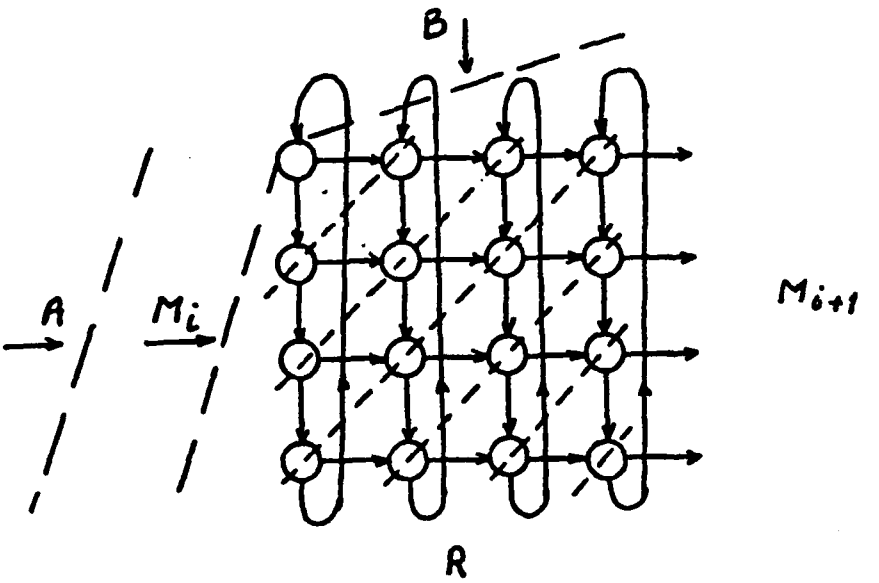


Figure 4: Systolic array for refinement procedure

SECTION 11

A STUDY OF PARALLEL ARCHITECTURES FOR IMAGE UNDERSTANDING ALGORITHMS

Dave Etchells

11.1 INTRODUCTION

The increasing maturity of Image Understanding (IU) software algorithms has placed growing demands on the hardware systems used to implement them. In addition, the time is rapidly approaching when these algorithms will find their place in real-world military and industrial systems. The speed, power, and size limitations imposed by such real-world systems will only further increase the demands on computer hardware.

It has been apparent for some time that conventional, single-processor, "Von Neumann" computer architectures are unable to meet the needs of advanced IU software systems, especially from the standpoint of providing real-time response within the power and size constraints mentioned above. In many cases, uniprocessor architectures are incapable of providing real-time response regardless of power and size constraints: The speed required of the arithmetic and memory units is several orders of magnitude beyond that foreseeable for any future circuit technology.

These limitations of uniprocessor machines have caused researchers to turn to multiprocessor, concurrent architectures as a solution to the demands of advanced IU processing. This expansion of the field of computer architecture has resulted in a proliferation of new computing structures, many of which are intended to match special processing requirements within the larger discipline of IU. [1-14] A key issue in the development of such machines is the degree to which they also meet the general requirements of the field, beyond the special needs which motivated their design and construction.

Comparative analysis of the new, concurrent architectures is complicated by the remarkable diversity which they encompass. This same diversity, however, makes the task of comparative evaluation all the more important. Various machines differ greatly in their utilization of hardware, making the selection of an optimal structure all the more important for future autonomous systems with limited space and power resources. To date, little attempt has been made to objectively analyze the performance characteristics of concurrent architectures, within the context of general IU processing requirements. The work that has been done in this area [15,16] has so far been limited in scope to just a few architectures, and primarily to biomedical image processing application areas. No concerted effort has been made to study the full range of parallel architectures, within the broader context of Image Understanding and scene analysis.

In our opinion, a primary reason for the lack of such comparative analysis is the absence of a set of widely accepted metrics for parallel hardware performance evaluation.

One of the goals of this report is to propose a set of common algorithms for use as performance evaluation standards within the field of IU. The selection of candidates for inclusion in the proposed metric set will be guided in large part by our view of anticipated future requirements of autonomous military systems.

The remainder of this report is organized into three principal sections. The first of these deals with general issues of performance evaluation, a system of classifying software algorithms, and an analysis of the processing requirements of common IU algorithms. The second section applies the results of the "software" section to a classification and study of existing and proposed families of concurrent architectures. The final, summary section of the report is a synthesis of information in the preceding sections. We draw conclusions regarding the strengths and weaknesses of existing families of computing structures, and suggest directions for future architectural development.

11.2 PERFORMANCE EVALUATION METHODS

In conventional numeric processing, there are two commonly used methods of performance evaluation. These are the instruction mix and benchmark program approaches. In the instruction mix approach, a set of programs is examined to determine the number of times each type of machine instruction occurs. The execution time of the programs in question on any particular processor can then be estimated by multiplying the execution time of each type of machine instruction by the number of times that instruction occurred in the benchmark. The sum of all such products, one for each type of machine instruction, may then be taken as an estimate of the execution time for the program set represented by the instruction mix.

The instruction mix approach is useful for rapidly evaluating the performance of conventional serial processors, but has little utility for the study of concurrent architectures. This is because of the importance of data movement in IU applications. That is, two algorithms might show identical statistics, in terms of the numbers of multiplications, additions, etc. that each require, but nonetheless exhibit radically different execution times on concurrent hardware, due to widely differing data movement requirements. For example, one algorithm might involve only data taken from a relatively small kernel, while the other might require global access to data scattered across the entire image data plane. Furthermore, in concurrent systems, the pattern of data movement is often at least as important as the amount of movement itself, since different architectures are able to take varying advantage of regularities in data movement. In addition to the importance of data movement, the instruction mix approach is unusable in IU applications because of the varying efficiencies in concurrent architectures. Many parallel processors, particularly SIMD arrays are not always able to use all of their available hardware to best advantage: Situations frequently arise in which a significant portion of the available hardware is idle, due to the lack of pertinent image data in the pixels associated with it. In such cases, a simplistic analysis based on aggregate instruction rates leads to performance figures substantially higher than can actually be attained. To have any meaning then, an instruction mix performance evaluation approach would have to include information on patterns of data movement, as well as some specification of the concurrency possible at each stage of execution. This last requirement is further complicated by the fact that different architectures vary in their ability to take advantage of parallelism inherent in software.

The benchmark program method of performance evaluation, on the other hand, avoids many of the problems inherent in the instruction mix approach just discussed. In this approach, a representative, or "benchmark" algorithm is programmed to run on a particular machine, and the actual execution time is measured. This execution time is taken to be characteristic of that machine for programs similar to the benchmark. If the benchmark program representative of the performance of the machine for actual applications. This is because such matters as data movement, machine efficiency, and the operating environment are naturally included in the final measurement.

The key problem of the benchmark program approach, of course, lies in the choice of the "representative program" that is used as the benchmark. Especially in the case of a field as broad as IU, it would be a hopeless task to map every interesting algorithm onto every proposed architecture. The problem of selecting characteristic algorithms is complicated by the breadth of the field, the wide range of approaches to any given IU sub-task, and by the fact that there are many areas in which there is currently no clear consensus as to the best algorithm for performing a given task. These are the parameters within which we must work. In the context of the current report, they are further modified by the need to arrive at a basis for meaningful comparison of architectures that does not involve explicitly coding algorithms to match individual architectures. These requirements lead us to seek the lowest level of program modules with the greatest degree of applicability across the entire range of IU algorithms.

In selecting representative algorithms or modules though, we must be particularly careful to not only represent the full range of application requirements (such as feature extraction, segmentation and classification, etc.), but to include as well the entire range of processing requirements, as seen by the hardware. In other words, while covering the entire range of IU algorithms, we must (since our objective is hardware evaluation) focus more on the processing load in making our selections, than on the overall structure or function of any particular algorithm. In order to do this effectively, we need to first develop a conceptual basis, or taxonomy, by which we might determine the unique processing requirements of each algorithm studied.

11.3 SOFTWARE TAXONOMIES

Little work has been done to date on the classification of software algorithms. Swain et. al. [17] proposed a six-point classification scheme which consisted of the following categories:

- o Type:
  - Enhancement
  - Extraction
- o Context Dependency:
  - Context Free
  - Context Dependent
- o Iteration:
  - Single-Pass
  - Multi-Pass
- o multivariancy:
  - Univariate Data

- Multivariate Data

o Execution Environment: - Real-Time
- Batch

o Computational Complexity: - n, n log(n), etc.

In this taxonomy, the "type" classification is based on the sort of processing that is being performed. Swain makes a distinction between those algorithms which modify the appearance or structure of an image (Enhancement), and those which evaluate that structure, to determine its contents (Extraction). "Context dependency" is a measure of the extent to which the processing performed is dependent on the image data values themselves. For example, an algorithm such as histogramming would be considered context-free, since the set of final values arrived at are solely a function of the values of the individual input pixels, independent of any relative associations that might exist between them. By contrast, an adaptive filtering algorithm would be context-dependent, since the output value (and, more to the point, the structure of the algorithm itself) for any given pixel would depend strongly on the values of the pixels surrounding it. The "iteration" category is self-explanatory: A single-pass algorithm only takes one pass over the input data to compute the output value, while a multi-pass one takes several. The "multivariancy" classification refers to the number of data values associated with each pixel of the input image. A simple grey- scale image would be classed as "univariate data," while a multi- spectral Landsat image would be considered "multivariate data." Swain's "time" parameter seems to be directed more toward the environment within which the software system will operate, rather than the structure of the algorithms themselves. While it is true that real-time algorithms are significantly different from those intended for batch use, this difference is a consequence of external constraints. That is to say that the classification of real-time vs. batch separates algorithms along lines dictated by the limitations of the hardware, rather than the sort of processing being performed. Swain's final software classification parameter is the "computational complexity" of the algorithms being considered. This refers to the relative dependence of the execution time of an algorithm on the size (n) of the data being manipulated.

These classification criteria focus more on the use to which various algorithms are put than on their structure. As a result, little explicit information is conveyed regarding the processing requirements of the algorithms so classified. While this approach results in a system useful to someone wishing to select an algorithm for use in a particular application, it has little use in the current situation, in which we wish to study algorithms to determine the unique demands which each places on hardware. We propose the following set of classification parameters as being more appropriate to the current effort.

- Functional Statistics.
- Local vs. Global.
- Kernel Size.
- Memory Intensive vs. Computation Intensive.
- Context Dependent vs. Context Free.
- Iconic vs. Symbolic.
- Object Oriented vs. Coordinate Oriented.

Here, "functional statistics" simply refers to statistics of the sort we referred to earlier in our discussion of the instruction mix approach to performance evaluation. In particular, we refer to the relative frequency of various arithmetic operations, such as addition, multiplication, division, etc. Such statistics are important in evaluating the performance characteristics of specific machines, but are less valuable in the study of generic architectures. This is because, apart from performance gains attributable to the level of parallelism employed, arithmetic performance is more a function of implementation than architecture. That is, the arithmetic performance of any architecture can be improved simply by adding (for instance) a faster multiplier, a special functional block for performing division or square rooting, etc. Such enhancements, while often significantly improving the performance of a particular machine, have little or nothing to do with the structure of the machine. Hence, our contention that such issues are implementation- rather than architecture-dependent. Under this category, we do not explicitly count data-movement operations. This is because data movement in concurrent architectures is highly dependent on the characteristics of the particular architectures in question. Since many machine structures (such as SIMD arrays) are able to take great advantage of regularities in data access patterns, a simple count of data values being accessed is not an accurate representation of the data movement required to effect that access (as would be the case in conventional serial processors). Data movement requirements are handled implicitly by this classification scheme through the subsequent "local vs. global" and "kernel size" categories.

The local/global distinction in the classification scheme is mostly a measure of the amount of a priori knowledge available to the concerning the location of data accessed by an algorithm. This in is contrast to the more usual interpretation of these terms as being indicative of the kernel size over which an algorithm operates. In the present usage, we consider an algorithm to have "global" scope if its domain cannot be restricted, a priori, to any definable subset of the image data. We therefore classify an algorithm as "global" if it may draw its data from anywhere in the image plane, regardless of whether it does so in all cases. This choice of interpretation for the terms "global" and "local" stems from the implications such access has for the architecture of individual processors within a concurrent architecture. If an algorithm may require the individual processors to have access to any portion of the image array, the architecture must provide for such arbitrary access in order to execute the algorithm efficiently. From the viewpoint of the computer architect, the fact that only a small number of pixels will be involved in a given operation matters less than the fact that those pixels may lie anywhere in the image plane. This is not by any means to imply that the number of pixels forming the kernel over which an algorithm operates is unimportant. Quite the contrary, the size of the "local" kernels processed by various algorithms can be quite important, particularly if they are of a size which exceeds the provisions of an architecture for "local" processing. As an example, some machines are optimized for the processing of kernels up to some maximum size (eg, 5 x 5 pixels). Manipulation of kernels larger than this upper limit can be highly involved and time-consuming. The maximum linear extent of a kernel can directly affect execution time on arrays employing nearest-neighbor communication. In our subsequent evaluation of various candidates for inclusion in our metric set, we will see that, while much of the raw computational load of IU algorithm would be one which requires the storage of a significant number of data values for each pixel of the image. An example of a memory intensive algorithm would be certain edge-detection routines, which

require the storage of edge magnitudes for each of several possible edge directions. Operations which involve sorting within a kernel, such as median filtering, require particularly large amounts of local memory. This is because all of the values found in each kernel must be stored and sorted for each pixel of the image. Even a relatively modest 5 x 5 kernel would thus require 25 words of local storage for each pixel of the image.

As with Swain's classification, we take context dependency to mean the extent to which the values output by an algorithm depend on relationships existing between input data elements. Note that this definition of context dependency does not refer to situations in which the output of an algorithm merely depends non-linearly on the input data values (as with thresholding). Such behavior is described independently by the linear/non- linear classification. Both context dependency and non-linearity are included in the more commonly employed term "data dependency." We have chosen to distinguish these cases as two separate classification parameters because of their different implications for hardware. Simple non-linearity, as exemplified by operations such as thresholding, usually involves only a comparison operation, and the setting of some sort of flag bit, based on the result. Context dependency, on the other hand, implies more involved processing, often predicated by a complex set of preconditions. Consequently, context dependency implies substantial differences in the program actually executed, depending either on the structure of the data, or on previously extracted information regarding that structure. Our usage of context dependency here therefore refers to a dependency of the program on the context of the input data.

The "iconic vs. symbolic" categorization is included because the two representations involve greatly different types of processing. In iconic processing, there is a direct relationship between physical storage locations and image pixels. Symbolic processing, on the other hand, involves the manipulation of lists, trees, and other data structures which contain image coordinates only as explicit entries in the data structure. Because of this difference of implicit versus explicit coordinate specification, architectures well suited to iconic processing often perform poorly when doing symbolic processing, and vice versa. The focus of the IU field historically has been directed more toward iconic processing. Consequently, most of the concurrent architectures built to date have been optimized for iconic applications. As the IU field matures however, attention is shifting from the simple, mostly open-loop classification and extraction that characterized earlier efforts to knowledge-based, intelligent systems that apply a significant amount of reasoning to the process of object recognition. As this trend continues to develop, symbolic processing will come to account for an ever-increasing amount of the IU processing load. We should accordingly expect to see more research activity in the area of architectures optimized for concurrent, symbolic processing.

The issues separating iconic from symbolic processing go beyond the simple dichotomy of the classification, however. While modern image understanding employs increasing amounts of knowledge-based symbolic reasoning, it nonetheless remains dependent on the lower levels of iconic processing for its raw information. Both sorts of processing are therefore important to practical applications. A consequence of this is that advanced IU applications almost invariably involve at some point the translation of data from the initial, iconic representation to a symbolic one amenable to the application of various rule systems and reasoning processes. Significantly, while we know fairly well

how to build machines that are capable of processing either iconic or symbolic data, no current architectures adequately address the problem of translation between the two domains. The difficulty of this translation lies in the fact that it is basically an object-oriented process. The data associated with a particular object might lie anywhere within the image plane, making it difficult to make any a priori assignments of individual processors to individual objects in a multiprocessor architecture. (Hence, most object-oriented applications are also classed as "global," according to our earlier definition of that term. Similarly, SIMD machines can only translate between iconic and symbolic representations one object at a time, due to their single instruction stream. The matter of how various processors handle object-oriented processing will be discussed in greater depth in the subsequent section of this report dealing with concurrent hardware. For now, suffice it to say that the problem of iconic to symbolic translation is a difficult one that as yet lacks adequate solution. For this reason, we have included "object oriented vs. coordinate oriented" in our list of software classification parameters. Coordinate oriented processing refers to situations in which the location of the data to be processed within the image is known in advance, independent of any characteristics of that data. On the other hand, in object- oriented processing, the location of the data to be processed is an implicit function of the data itself, and of relationships existing within the data Object-oriented processing commonly encountered in IU applications includes such feature-extraction operations as boundary tracing of objects, computation of medial axes or generalized cones, etc. Such operations typically involve following object features on a pixel-by-pixel basis. At the same time, the pixels associated with an object may lie anywhere at all on the image plane. Few architectures provide for such image-wide access while simultaneously allowing independent processing of various parts of the image. As mentioned above, we will be covering this issue in greater detail as part of the discussions relating to each of the processor categories studied in the hardware section appearing later in this report.

This list of software classification categories provides the basis for a study of algorithm characteristics, distinguished by the demands they place on the processing hardware We will use this taxonomy in the following section on IU processing requirements, and again in our overview of contemporary and proposed concurrent architectures.

11.4 IU ALGORITHM OVERVIEW AND METRIC SET

As mentioned in the introduction, one of our goals in this report is to develop a set of common algorithms which will serve as performance evaluation standards within the IU discipline. In this section, we briefly review the most common types of processing encountered in image understanding, and select a set of algorithms for inclusion in a metric set which most accurately represent the range of IU processing.

Before launching directly into this discussion, though, it is first appropriate to consider the level of algorithms that would be most profitable to study. We would like to find algorithms or operations which enjoy wide application across the entire IU field. We must balance this desire against the requirement that the algorithms selected be uniquely representative of the requirements of IU, as identified by our taxonomy. Obviously, great commonality of application can be found if one examines the very lowest level of operations possible within an architecture. That is, virtually every IU algorithm in existence

uses the operations of addition or multiplication at some point in their execution. Simple addition and multiplication however, have very little about them that is uniquely characteristic of the requirements of IU. On the other hand, we could select a particular algorithm developed by a particular researcher that would contain much more of the essence of IU processing, but that would be rather limited in application.

We suggest that algorithms or, more properly, sets of operations can be found that are intermediate in level between the two extremes just mentioned. Such "unit operations" as we call them function at a low enough level that they may be found as functional blocks within a wide range of more highly developed image understanding application programs. At the same time, they are of a sufficiently high level themselves that they embody characteristics, as defined by our previously discussed taxonomy, that uniquely define the processing commonly encountered in IU systems. Table I lists the unit operations that we have selected for consideration, and shows how they fit into our classification scheme. A discussion of these unit operations and their classification follows.

In the following discussion, it is important to note that many of the unit operations described can be used in ways contradictory to their primary classification. (For example, almost any coordinate-oriented procedure can be applied to the data representing only one or more unique objects. The resulting program would then most properly be object-oriented.) Such application does not in any way invalidate their selection as part of the metric set, based on our classification of them. This is because our intent here is not to rigorously classify the algorithms, including all variations of their usage. To the contrary, we only wish to insure that we have adequately accounted for the various types of processing, represented by our taxonomy, that are typical of IU. Thus, as long as we include examples of both coordinate- and object-oriented processing, we care little if our examples of either type may sometimes be used in a context opposite to that of our primary classification.

Thresholding was selected as the first candidate for inclusion in the metric set. It was chosen because it is the simplest example of a parallel, non-linear operation, and because of the wide application it finds in IU processing. It is further classified within our software taxonomy as being local, because it by definition takes as input only the values of individual pixels. Thresholding may be either context-free or context-dependent, depending on whether the threshold value is selected adaptively or not. If the threshold value is the same for all pixels in the image, the operation is context-free. On the other hand, if the threshold value is set locally, as some function of the local data values (other than the threshold value itself), the operation is context-dependent. Since either type of thresholding does not typically involve the storage of intermediate values, little local memory is required, and it is classified as computation intensive. It is also coordinate-oriented, even in the context-dependent case, because the data required to generate a given result always lies within a small area surrounding the pixel being processed. Likewise, the operation is strictly iconic.

Context-free thresholding is widely used in IU for such tasks as thinning, and for identifying regions or points of interest. Context-dependent thresholding can be found in routines doing adaptive filtering or connectivity linking. In such situations, the threshold

value is adjusted depending on the values of adjacent pixels or some properties of the local ensemble. Because the two types of thresholding stress machine architectures differently, it would be wise to include examples of both in the metric set. The context-free example would be quite simple to implement: All pixel values are compared with a single, arbitrary value, and either the contents of pixels containing values less than or equal to the threshold value are set to zero, or a flag is set in all such pixels. The action taken would depend upon the architecture being examined: Some machines handle data selection by toggling condition flags, others by simply zeroing (or otherwise modifying) the contents of registers associated with the affected pixels. We suggest the use of a "less than or equal to" condition, rather than the simpler "less than," because of the different ways that various architectures handle compound conditionals. Some processors allow the setting of multiple condition flags (or their equivalent) with a single instruction, while others require that two separate comparisons be performed, and the results combined in a third operation. Obviously, the former solution is the more desirable of the two. The requirement for such compound conditionals appears frequently, either in the form just mentioned, or in the need to test for a "not equal" condition, which may be expressed in some machines as a "greater than or less than" condition. The context-dependent case could be implemented by requiring that the threshold value be determined for each pixel in the image as the average value of the 3x3 neighborhood surrounding it. This would test the ability of the architecture to compute and load data values for context- dependent processing, based on local data characteristics. The 3x3 average would overlap somewhat with the subsequent metric candidate of convolution, in terms of data movement and computation requirements. Since convolution usually involves a larger kernel, as well as different memory requirements (for storage of the weighting function), the redundancy in the metric set resulting from the inclusion of the 3x3 averaging here is negligible.

We have chosen convolution as the second member of our metric set. Convolution is widely employed in filtering functions such as edge detection and enhancement [18-20], and as part of such procedures as connectivity linking and region growing [21-23]. It is basically a linear, arithmetic process, in which the data values within a local neighborhood are individually multiplied by a set of weight values, and the resulting products are summed to produce the final result. Such a sum-of-products is computed over the local neighborhood of each pixel of the input image. As just stated, convolution would be classified in most cases as a linear, local operation. In some situations, the output is made non-linear, but this typically occurs through a thresholding operation, which we have already included in the metric set. In most cases, convolution is also context-free, with the weighting functions being invariant across the image. In some forms of adaptive filtering, a multi-pass iterative technique is used, with the local weighting functions being modified by the results of the earlier pass. An example of such usage would be an algorithm to extract the lines forming loops and whorls of fingerprints. In this application, the weighting values of a line-enhancing filter are modified according to the dominant local line direction found on a previous pass. Such usage is obviously highly context-dependent. Context-free convolution by itself is computation- rather than memory-intensive. Some of its applications do involve the storage of intermediate products, however. Edge-detection, for example, usually involves a series of convolutions, one in each edge direction being tested for, with the intermediate results of each individual convolution being stored for subsequent comparison and selection of the

largest directional value at each point. We consider this process of selection from among multiple values to be an example of sorting, though, which we treat as a separate member of the set of software metrics. Unlike context- free convolution, context-dependent convolution can involve substantial amounts of local storage, depending on the architecture in question. This is because some machines, particularly those having a cellular architecture, require that the various weighting coefficients for each of a range of possible convolutions all be stored in the local memory. For this reason, it would be advisable to include examples of both context-free and context-dependent convolution in the metric set. Both types of convolution are strictly coordinate oriented, and operate within iconic representations.

Another issue in the selection of examples of convolution for inclusion in the metric set is the size of the local neighborhood, or kernel, that is involved in the computation [24,25]. Kernel sizes have tended to increase proportional to the sophistication of the IU algorithms employing them. Early edge operators, such as the Sobel, operated on kernels of 3 x 3 pixels. Most modern edge algorithms employ 5 x 5 kernels, and some algorithms use kernels as large as 17 x 17. To some extent, the size of the kernel can be thought of as being representative of the amount of knowledge that the program has about the structure for which it is searching. The 5 x 5 kernels that are now common can only be expected to grow in size as time progresses. Kernel size is important in that it can affect the execution time of an algorithm non-linearly on certain processors. This is especially true in some machines optimized for local-neighborhood operations. Such machines often have special hardware for performing convolutions that can only accommodate kernels smaller than some maximum size. Kernels smaller than or equal to the maximum size designed for are all processed with roughly equivalent speed. Larger kernels usually reqf unique weight values present. This is because the difference in execution speed between having to perform an additional subtraction operation (rather than simply summing positive values) is negligible in virtually all architectures, compared to the time required to perform an extra multiplication. The most popular algorithms using convolutions seem to employ about N unique weights in their N x N kernels. That is, a typical 5 x 5 kernel uses about 5 or 6 unique weight values.

From the preceding discussion, it would seem that the convolution metric should include the following: 1) A simple, context-free convolution with a 5 x 5 kernel and perhaps 5 different weight values; 2) A context-free convolution with a 17 x 17 kernel and 20 unique weight values; and 3) A context- dependent convolution with 5 x 5 kernels, where the processing occurs in two passes: In the first pass, each of 6 different kernels is applied, and the results stored. A new image is composed of the set of maximal magnitudes resulting from the first-pass convolutions. Then, in the second pass, the kernel that resulted in the largest average magnitude on the first pass for each pixel is applied to the new, maximum-magnitude image, centered on the pixel for which it produced the maximum magnitude. This last test is artificial in that no algorithm of which we are aware employs exactly this sequence of operations. It does serve the purposes of the metric set quite well, however, in that it exercises the local storage capabilities of the architecture under test, by requiring the conditional selection of kernel weight factors, based on the results of earlier processing.

The third class of algorithms we have considered for inclusion in the metric set are

those which perform sorting operations. Sorting is a local, non-linear, memory-intensive process. As mentioned above, it often finds application in conjunction with convolution, in operations such as line-finding, where the largest of several values must be selected. In this form, it has already been included as part of the third case of convolution processing, discussed immediately above. Another application, requiring more memory than the preceding case is median filtering [26]. Median filtering involves the selection of the median value from among a number of pixel values found in a local neighborhood. Median filtering is often used for size discrimination and connectivity processing. It is useful for these purposes because objects smaller than the kernel used are removed from the image, with relatively minimal additional disturbance to the image. Sorting in general is more purely context-dependent than any other algorithm we have examined thus far, in that the shuffling of pieces of data, or the setting of pointers and flags is strictly a function of the data values involved. While it may be employed in either a coordinate- or object-oriented fashion, its usual usage is in a coordinate- oriented mode. It may also occur in either iconic or symbolic representations, depending on the application.

The example of sorting we propose for inclusion in the metric is a median filter operating on a 5 x 5 kernel. This will require the sorting of 25 data values for each pixel of the input image, and should fairly represent the processing load of many memory-intensive algorithms. This particular example is strictly coordinate-oriented and deals only with iconicly represented data.

We have chosen histogram generation as the fourth member of the set of software metrics. A histogram is simply a set of numbers indicating the frequency of occurrence of each of a set of intensity values or intensity ranges within an image. Histogramming is representative of what might be called "statistical processing," and finds wide application within IU. The popular region-splitting segmentation algorithm [27-29] uses histogram characteristics as the criteria for distinguishing distinct regions within images, and many object classification algorithms use intensity histograms to classify and distinguish target objects. The processing requirements of histogram generation differ from those of the operations discussed so far in that histogram computation operates globally in a context-free fashion. It is most properly thought of as computation- intensive, according to our previous definition of that term, since the memory required for storage of intermediate results is quite modest: Only one memory location is required for each value being tallied for the histogram, regardless of the size of the array being processed. On the other hand, the actual execution of the algorithm is memory access-intensive, particularly on MIMD and pipelined machines. This is because the memory locations used to store the individual tallies are accessed repeatedly as the tallies are updated for each pixel processed. On cellular machines, the global nature of the processing places special demands on the architecture, particularly in systems employing nearest-neighbor communication exclusively. If proper provision is made for global communications across the array though, cellular machines can be highly efficient for this sort of processing [30].

Histogram computation is also linear with respect to the input values, and is most often applied within an iconic context. While we have classified histogram processing as coordinate- oriented, it should be noted that many applications use it in an object-oriented context. An example of such usage would be the case, mentioned earlier, of target-identification algorithms, which frequently require the computation of intensity his-

tograms for each of a class of objects present in an image. While this object dependency is not an intrinsic characteristic of the histogramming process, it does represent an important mode of usage of the histogram operation. Furthermore, this object- oriented usage can have significant implications for the execution of the operation on various architectures. In particular, the algorithm cited above [30] for histogram calculation on cellular arrays is most suited to the calculation of global histograms. The computation of individual histograms for each of several objects in the image would require separate passes for each object evaluated. For this reason, it would be important to include examples of both global and object-oriented histogramming in the metric set.

Our recommendation for the histogram generation metric is as follows: 1) A 256-level (8 bit) histogram computed over the entire image; and 2) An 8-bit histogram computed for each of six objects (an arbitrary number) in the image. The six "target" objects would be identified on the image by the presence of "1s" set in a binary mask having the same resolution as the source image itself. The object-oriented histogram generation task would consist of producing an 8-bit intensity histogram or each object identified by a discrete pattern of "1s" in the binary mask. A part of this task would obviously be to identify and separate the regions corresponding to each of the objects. This portion of the histogram generation metric would provide the strongest example of object-oriented processing we have encountered so far in the metric set. It might, in fact, be informative to require that the execution time of the object- separation sub-task be separately quantified, and used as an independent measure of the ability of the architectures tested to perform object-oriented processing.

Correlation operations were chosen as the fifth metric set class because they involve local processing with a high degree of context dependency. As typically applied (in stereo processing), portions of one image are compared against (correlated with) various regions of another picture [31-33]. The correlation process is basically a convolution, but with the weighting functions being the data values of the reference image, rather than some externally generated filtering function. This process of data-dependent convolution can be conceived of as a matched filtering operation, where the reference image is the object for which the filter function is optimized. Correlation of this sort thus tests the ability of an architecture to rapidly access different sets of weighting values for convolution processing As mentioned above, processing of this type is very common in both simple intensity-based and more advanced feature-based stereo algorithms. Correlation is a linear process, most frequently operating in the iconic domain, although some approaches operate at least partially in a symbolic fashion [34]. It is also a memory-intensive operation, in that correlation coefficients are typically evaluated and stored for each pixel of the image for each of several pixel displacements (between 3 and 15). The correlation coefficients for each pixel are then compared to each other, and the largest chosen as that corresponding with the most likely relative displacement between that pixel and the equivalent one in the reference image.

As most commonly applied, correlation for stereo vision is performed only in areas containing features of interest, such as regions of high contrast, high-magnitude edges, or corner elements. These areas of interest may lie anywhere in the image plane, and so the processing involving them could properly be thought of as object-oriented. On the other hand, this object- orientation is not usually an intrinsic part of the algorithm itself,

but more often represents an attempt to reduce the computational load on the conventional serial computers used to develop and test the algorithm. Since such computational load reduction is rarely necessary (and frequently undesirable) on highly concurrent machines, we have classified correlation processing as coordinate-oriented.

We propose the following as the correlation processing metric: Two images would be provided, representing two stereo views of the same scene. The objective would be to determine the most likely relative displacement between pixels in the "right" image and those in the "left" image. The most likely relative displacement would be that which produced the maximum correlation coefficient over a 7 x 7 local neighborhood of the pixel in question. For the purposes of the test, it would be assumed that the two cameras had identical optics, and were mounted on a common centerline that was parallel to the "ground" plane of the scene represented. Thus, there would be no need to correct for scale, rotation, and tilt factors. Relative pixel displacements ranging from 0 to 6 pixels would be evaluated for each pixel of the right image.

We have chosen interior point selection as the sixth member of our proposed metric set. Interior point selection is the process of identifying those points of an image that lie within closed boundaries defined by previously located lines and edge segments A point is typically determined to be on the interior of a closed boundary if there are edge segments within a certain radius of it, with the proper direction, and in a majority of the directions checked. More sophisticated algorithms may impose the further constraint that an interior point be bounded by a pair of edge elements of opposite polarity (ie, light/dark and dark/light). Interior point selection processing is non-linear, because of the binary nature of the output data (a point is either inside a boundary or not). It is also most properly classified as global, since the processing is performed for all pixels in the image. Some implementations might be considered to involve only local processing, due to restrictions on the size of the neighborhood searched for pertinent edge elements, but on the whole, the global classification is most appropriate. Interior point selection most naturally operates on iconically represented data, and is computation- rather than memory-intensive, in that little intermediate data is stored for each point evaluated. The operation is also obviously context-dependent, according to our earlier definition, and is object-oriented in most implementations.

This classification of interior point selection touches upon an important and difficult issue in the description of IU software. The problem lies in the fact that the structure of the hardware being used to implement an algorithm often has a substantial impact on the nature of that implementation. We saw this to some extent in our previous discussion of correlation processing, which is often performed on serial machines in an object-oriented manner, focussing on particular "points of interest" in the image, in order to minimize the amount of processing required to locate objects in three-space. The issue is raised again by our classification of interior point selection. Most current implementations of such algorithms operate in an object-oriented manner, by applying some sort of preliminary selection criteria to produce likely candidates for interior points, subsequently performing the detailed selection processing only on the likely candidates. The process is again an attempt to reduce the computational load to a minimum for serial processors, and is typical of similar techniques which appear throughout the field of IU. On many concurrent architectures though, little time is saved by such "planning" or pre-selection processes.

This is because, on such machines, it is just as fast in many cases to process the image homogeneously, rather than to separate out certain parts of it for special attention. Thus, while algorithms developed on and implemented for conventional serial processors may be object-oriented, equivalent algorithms implemented for concurrent machines may process the data uniformly, in a coordinate-oriented fashion. The classification process is further complicated by the fact that many distinctions between algorithms are of a quantitative, rather than qualitative nature. The ultimate solution to this issue is beyond our present reach, so for the purposes of this report, we are taking the approach of: 1) Inserting this caveat; and 2) Being guided in our classification by the currently dominant modes of usage of the various algorithms, making note of potential differences in classification that might arise from implementing the algorithms on different architectures.

Returning to our discussion of the interior point selection metric then, we propose for that metric the following: An edge image, indicating edge magnitude and direction for each pixel, representing a scene containing five or six clearly defined objects of varying shape and dimension, appearing as light objects on a dark field. The process used to select interior points would be: 1) Scans would be taken across the image along each of three possible edge axes (total of 6 possible edge directions). For each scan direction, regions between edge segments antiparallel to within 120 degrees of each other would be "marked" as interior point candidates. 2) For each pixel, the number of candidate "marks" that it had received in part (1) of the algorithm would be counted. 3) The mark counts of part (2) of the algorithm would be thresholded. Points having two or more "marks" would be considered to be interior points.

As with the other metric set algorithms proposed earlier, we must emphatically state that the proposed algorithm is not intended to represent the state of the art in the particular area it addresses (in this case, interior point selection). The proposed algorithms are not even intended to be particularly effective at their purported tasks. Our intent is solely to provide an easily-coded algorithm which has processing requirements representative of certain classes of IU algorithms.

We have selected line-finding as the seventh member of the IU metric set. By "line-finding," we mean those routines which are concerned with linking edge segments together into lines, and "tracing" the resulting lines to determine their lengths and orientations. This definition differs from common usage, in which the term "line-finding" is applied to complete algorithms which locate and evaluate edge elements, in addition to performing the linking operations which we are considering here [20,35,36]. This is by far the most clearly object-oriented process that we have examined thus far, in that the "tracing" operation necessarily involves following the line wherever on the image plane that it might go.

Line-finding is particularly interesting, because it operates on iconically represented data, producing as output information that is symbolically represented. In other words, it proceeds from an image of a line to a list of the characteristics of that line. As mentioned earlier, and as we shall see in our subsequent discussion of machine architecture characteristics, such translation processes pose particularly difficult problems for computer architects. In fact, no current architecture adequately meets the requirements of such algorithms. The detailed reasons for this will be left for the hardware section of this

report, but basically involve a conflict between memory requirements, communication capability, and data contention among the processing elements of concurrent architectures. This poor match between any existing architecture and the requirements of the problem also makes the classification process difficult. On array machines, the process is computation-intensive according to our earlier definition of that term, because the generated information about the lines being traced either remains distributed across the memories between processors overloads the interprocessor communication network. Line-tracing is otherwise classified as non-linear, due to the binary decisions as to whether a pixel is or is not on the line being followed; global, because there is no a priori knowledge of where any particular line might go, and therefore no information regarding the extent of memory access required for the local processor to follow it; and context-dependent, for reasons too obvious to mention.

We propose that the line-finding metric consist of the following: The input image would be an edge image, containing edges in six possible directions, of varying intensity. These edges would have been "thinned" previously, to leave only those which were strongest, and most likely to be associated with extended linear structures in the image. The earlier steps of most line-finding algorithms employ methods for executing such thinning, and the test imagery could be prepared with such an algorithm. Given this image, the operations performed would be to: 1) Examine the 8-neighbors of each pixel, on a 3 x 3 grid to determine the predecessors and successors. Only the neighbors in directions approximately parallel and antiparallel to the central pixel would be examined. (We will take discontinuities of edge direction to mark the end of one line and the beginning of another. 2) A neighboring pixel is linked to the central one if their directions are within 30 degrees of each other. In the case of more than one neighboring pixel satisfying this criteria, a decision function is calculated for each pixel, taking into account the positions of the two pixels relative to the direction of the central one being linked to, and the relative magnitudes of the candidates. The candidate having the highest value of this decision function would be selected for linkage. 3) Once links have been made to all possible predecessors and successors, the "boundary segments" are traced. This procedure, taking the predecessor/successor data and using it to construct linked lists of edges corresponding to linked lines is the heart of the line- finding metric. Processing for this step occurs in two passes. In the first pass, tracing begins with edge points having no predecessors, continuing for each segment until an edgel is reached that has no successors. The second pass begins arbitrarily at any point not traversed by the first pass, and proceeds in the same fashion as the first pass. (Note that this simple-minded algorithm creates two separate lines at any point where there is a "fork" in the line. (Ie, fork "branches" are not shown as attached to their parent "limb".) This somewhat simplifies the implementation of the metric, without significantly affecting the processing requirements of the task. The end result of the line-finding metric will be a file of line segment descriptors, consisting of starting coordinates, a linked list of successor and predecessor pixels, and a pixel count, indicating the length of each segment. The task would only be considered complete when all of the segment descriptors had been transferred to the host processor, preparatory to recording into mass storage. (This line-finding algorithm is a simplification of the one developed by Nevatia and Babu [20].

Shape descriptions were chosen as the eighth member of the metric set because they also involve the translation of information from the iconic to the symbolic domains.

The difference between shape description processing, and that associated with line-finding is that shape description algorithms typically involve the "tightly coupled" transmission of information across somewhat greater distances than does line- finding. While line-finding is strongly object-oriented, it is feasible in most cases to sub-divide the image into smaller regions for processing by individual execution units. In those instances where a line segment being traced extends beyond the boundary of the area represented in the memory of the local execution unit, the "entry point" and direction of the line can usually be passed to a neighboring processor into whose domain the line extends, so that that processor may continue the required processing. This is usually not feasible in shape description algorithms, such as medial axis transforms [37] or generalized cones [38,39]. The reason is that these algorithms locate the centerline of objects by taking multiple "scans" through the image segment in question, perpendicular to the iteratively calculated axis of the object. The midpoint of the object at that point along its axis is taken to be midpoint of the path traversed through the object. If an object being examined in this manner straddles the subdivision boundary between two adjacent processors, the resulting message-passing requirements would quickly overload any conceivable communications network. Such a high communication bandwidth is required in these cases because path length and direction information must be passed for every scan line crossing the image subdivision boundary. This might involve the passing of a separate message for each pixel along that boundary. It is because of this high, local loading of communication networks in multiprocessor architectures that we have included shape description processing in the metric set. As to the other categories in our classification taxonomy, we categorize shape description processing as global, context-dependent, and computation intensive. It should also most properly be considered linear, since the output data depends linearly on the shape of the object being examined.

Our metric for shape description would consist of executing a generalized cone transform on an input image containing an irregularly shaped object, such as a humanoid doll or airplane. The single object would be sized to substantially fill the frame, to maximize communication loading in multiple instruction stream architectures.

Our final two entries in the proposed metric set are examples of more purely symbolic processing. Graph matching, the first symbolic metric involves searching a graph for a sub-graph having a particular, specified structure. The difficulty of this sort of processing lies in the fact that the computational complexity grows exponentially with the size of both the graph being searched, and the sub-graph being searched for. The key issue in implementing such algorithms on concurrent architectures is the partitioning of the problem among available resources. Our graph-matching metric would involve searching a graph of 100 nodes for a sub-graph of 10 nodes.

The last metric candidate is symbolic prediction, involving the application of a set of rules to a set of existing data to predict the probability of occurrence of some particular condition. Such tasks are commonly implemented as "production systems," where the "triggering" of a rule causes some modification to the body of data being operated on (either addition of data, deletion of data, or a change of value for some element of the data-base). Rules usually have an "if-then-else" form, where various actions are taken on the knowledge base determined by the presence or absence of certain conditions. As with graph matching, the key issue in implementing such algorithms on concurrent

machines is that of partitioning. The problems lie in the division of the knowledge base between the available storage regions, and the links (via the rule system) between the data stored in these regions. (In this usage, a "storage region" would most commonly be the local memory associated with each processor in a concurrent system.) As with many of the object-oriented processing we studied earlier, the problem focusses on the bandwidth of the communication network, and partitioning the data and program elements so as to minimize the bandwidth requirements.

Our suggestion for a metric in the area of symbolic prediction would be to abstract a representative piece of predictive processing from that performed by an existing system such as ACRONYM. Such an approach would have the advantage of providing a ready comparison with many existing, serial machines.

11.5 CLASSIFYING SYMBOLIC PROCESSING

It is somewhat apparent from the above that symbolic processing does not fit easily into our classification scheme. This is because virtually all such computation is described similarly within the taxonomy. (Eg, as non-linear, memory intensive, object-oriented, and context-dependent.) This lack of distinction between various symbolic processing tasks is reflected in the fact that both of the operations we have selected as representative of symbolic processing are quite high level, involving a wide range of processing requirements. This contrasts sharply with our selection of "unit operations" to represent the processing loads imposed by more iconic processing. It would thus appear that an extension to our classification method is in order, to allow us to identify "unit operations" within the symbolic regime. We have not done this as of this writing, but a paper by Hillis [40] indicates four possible computational categories which might be used to describe symbolic processing. Hillis' list of critical operations for symbolic computation may be paraphrased as follows:

- Deduction of facts from semantic inheritance networks.
- Matching of patterns against sets of assertions, demons, or productions. Best matches must be selected in the absence of a perfect match. ("Fuzzy" decisions.)
- Sorting of sets according to chosen parameters.
- Searching graphs for sub-graphs with a specified structure.

Our proposed graph matching metric directly addresses the fourth of Hillis' categories, while our prediction metric is more generally directed at the first three areas. Further study in this area should permit the development of more refined metrics which more specifically target the processing requirements of each of these four areas.

11.6 HARDWARE ANALYSIS -- INTRODUCTION

In the preceding sections, we established a basis for understanding the different processing requirements of IU algorithms. We have used this taxonomic basis to study the differing requirements of a range of typical "unit operations" commonly employed in the IU field, and selected a number of these operations as candidates for a software metric for the evaluation of concurrent hardware performance.

The software taxonomy also provides a basis for a more general discussion of the characteristics of various classes of concurrent architectures. We can gain a relatively complete understanding of the generic capabilities of different machine classes by examining their performance in each of the categories included in our earlier taxonomy. The purpose of this portion of the report is to perform just such an analysis.

Our approach here will be to divide the entire range of concurrent computer architectures into nine categories. The categories used are: cellular numeric, pipelined, multiple instruction stream (MIMD), number theoretic, systolic, hierarchical, "broadcast," data-driven, and associative. Within each category, we will first present the general structure of machines within the class, followed by one or more examples of specific implementations or proposed implementations of the architecture, and will conclude with a discussion of how well machines of that general type perform the various sorts of processing represented by the categories of the software taxonomy.

11.7 CELLULAR NUMERIC ARCHITECTURES

Cellular machines are those composed of an array of identical processors, or cells, directed by a common instruction stream, but operating on separate data. Image processing is usually performed by assigning pixels to processing elements of the array in a one-to-one mapping. In most cellular architectures, the program interpretation and sequencing is performed by a single "control processor" (CP), separate from the array elements: The array processing elements are typically only capable of executing arithmetic and logical instructions under the direct control of the central CP, although most implementations do provide for some data-dependent data selection through the use of "masking" operations.

Data flow within cellular machines is primarily between adjacent processing elements, using nearest-neighbor links. Some machines also permit the "broadcasting" of values along the rows and/or columns of the array, or to all elements of the array simultaneously. Such broadcast or replication capabilities greatly speed the execution of certain algorithms. Provision is also usually made for data values to be passed between the CP and the array elements, for such purposes as thresholding and data- dependent program branching.

Cellular machines are perhaps the most popular of all concurrent architectures for image processing work, with quite a number having been built to date. Machines built or proposed to date include the Goodyear MPP (Massively Parallel Processor) [41], the CLIP at University College, London [42], the ICL DAP [43], the Hughes 3-D machine [44], the BAP [45], and the Illiac IV at the University of Illinois.

11.7.1 Characteristics of Cellular Architectures

Cellular machines have a number of hardware advantages over other types that make them particularly attractive for VLSI implementation. Their extreme parallelism permits the use of serial arithmetic in the individual processing elements, while still achieving very high throughputs. Circuitry for serial processing is in turn both simple and com-

pact, permitting high cell densities in the completed machines, and high yields of the individual cells. Furthermore, the high regularity of the array hardware gives a high ratio of replicated-to-designed circuits for such systems when CAD is used in their layout. Finally, since communication among array elements is primarily between nearest neighbors, the signal routing requirements are relatively modest.

Cellular numeric arrays are often easier to program than other types of machines because of the natural match between the structure of the computing array and the structure of the pixel arrays being manipulated. This topological correspondence eliminates much of the address calculation overhead required by architectures lacking such a match, and furthermore eases the visualization required of the programmer in the implementation of various algorithms.

The nearest-neighbor communication scheme employed in cellular architectures means that they are most efficient when executing programs which are local, according to our earlier definition of that term. Global operations requiring a great deal of data sharing between processing elements separated widely in the array are usually less efficiently executed. Most implementations, however, allow constants to be rapidly broadcast across the entire array, for use as thresholds or multipliers in the local operations. This capability is particularly useful for such operations as histogramming (one of our software metrics), where a large number of values must be compared with every element in the array.

If provisions are made for data-dependent masking operations within the array elements, cellular machines are about equally effective in performing linear or non-linear operations. In any given application, a particular machine may be more or less efficient at non-linear selection operations, depending on the amount of storage provided for different mask vectors. Some machine provide only one or two bits of mask vector storage at each cell site, while others permit any of the general registers of the machine to be used for such storage.

Most cellular implementations proposed or constructed to date have relatively limited amounts of storage available in the individual elements of the array. This means that they are less well suited to memory intensive problems than they are to computation intensive ones. This limitation is not inherent in cellular architectures though, but rather is a reflection of technological limitations in force at the time these machines were developed, as well as conscious design choices made by the designers of the machines. Cellular machines, to a somewhat greater extent than is the case with other architectures, present the designer with a tradeoff between either increasing the amount of local memory available, or fabricating a larger array. Since most IU problems are far more computation intensive than they are memory intensive, most designers opt for a the largest array size possible with the technology available, at some expense in the amount of memory present at each cell site. Thus, while cellular architectures are typically less suited to memory-intensive computation, this need not be the case, although the above- mentioned design considerations will probably result in the persistence of this characteristic into the foreseeable future.

The single instruction stream of cellular machines means in most cases that they are less efficient in context-dependent than in context-free applications. In most cases, context-dependent programs are executed exhaustively for all possible cases at all array sites. That is, if there is a program branch in which data each object or structure in the image must be processed separately and in sequence. This is slow and, since the processing is usually concentrated in only a small region of the total array area, rather inefficient as well. As we shall see, though, few architectures perform well in this respect.

Finally, coming to the last taxonomic category, cellular numeric arrays are almost exclusively oriented toward iconic processing. Some thinking has lately been directed [46] toward providing the individual processing elements with an associative capability, which would extend their capabilities into the symbolic domain as well. This appears to be a promising approach, but the work done in this area is so far very preliminary.

11.8 PIPELINED ARCHITECTURES

Pipelined machines consist of series of processing elements arranged in chains or "pipes," such that each element in the pipe accepts data from its predecessor, and passes results to its successor. Processing occurs simultaneously in all elements of the chain, with the data and results flowing down the chain as liquid through a pipe. In addition to the concurrency achieved by having processing occur simultaneously at various points along the pipe, some implementations employ a multiplicity of pipes as well.

In contrast to cellular arrays, pipelined hardware usually handles its own program interpretation and sequencing, rather than relying on a separate controller for those functions. Also in contrast to cellular machines, each pipe of a multi-pipe machine usually has access to the entire image array. On the other hand, functional units within the pipe typically only access data through the hierarchy of the pipe itself; ie, from their predecessor in the chain. This places some constraints on the patterns of data access permitted by the architecture.

Because of their dependence on temporal relationships within the algorithms implemented on them, pipelined architectures are for the most part a good deal more difficult to program than conventional machines. This is because the programmer must understand the structure of the pipe in detail, and organize his program to take advantage of the "sequential concurrency" built into the hardware. For this reason, most commercial pipelined machines come with very extensive software libraries so that most users need not directly involve themselves with the hardware. This of course is a compromise, and one that is only successful if the algorithm to be programmed can be composed from the standard elements available in the library supplied.

Pipelines are the most popular concurrent architecture in the commercial marketplace, with most of the so-called "array processors" available on the market actually having pipelined architectures internally. (These machines are usually sold as as attached processors for standard minicomputers.) Part of the reason for this commercial popularity is probably the fact that pipelined architectures are rather easily built up out of

standard MSI and LSI circuits, in contrast to more highly evolved architectures which require custom VLSI for their efficient implementation. This is not to suggest though, that pipelined machines lack potential for application within the IU community. To the contrary, a number of research machines have been proposed or built with pipelined architectures, including DIP [47], GOP [48], the "CYTOCOMPUTER" [49], and an interesting "pipelined array," combining some of the characteristics of both pipelines and cellular machines [50].

11.8.1 Characteristics of Pipelined Architectures

Since a single pipeline must naturally have access to the entire image array, pipelines are by nature equally suited to both local and global processing. In multi-pipe machines, this characteristic may be somewhat modified by the manner in which memory is distributed between the separate pipes. If the individual pipes draw their data from a single, common memory, the resulting machine may suffer from memory contention between the execution units. To eliminate this problem, the designer of such a machine might opt to have separate memories for the individual pipes, segmenting the image between them. If this is done though, the machine loses some of its generality, and performance on globally-oriented algorithms is affected. Of course, the brute force approach of replicating the entire image in memory as many times as there are pipes may be taken. This can be extremely costly in terms of memory space though, and the problem still remains of passing information and results between the pipes, should that be required.

Pipelined machines also usually perform both linear and non- linear processing equally well, since non-linear operations only require the addition of comparator circuitry at appropriate locations along the pipes. Context-dependent processing is quite another matter, though. Since the concurrency exhibited by pipelined machines derives from certain fixed temporal dependencies in the program, as well as on fixed sequences of operations, any disruption in the normal flow of data is highly deleterious to the overall performance of the system. Thus, such machines are quite efficient when the type and sequence of operations to be performed is rigidly determined. This is the case in many lower-level operations such as thresholding and convolution. Speed obtained in this manner carries a penalty though, in that there is a fixed delay from the time that data is first presented to the beginning of the pipe to the time that the first results appear at the output. This delay corresponds to the amount of time that it takes for data to pass through all the functional units along the pipe. Because of this, each context- determined program branch will incur a fixed "pipeline delay," while the results from the new branch move down (fill) the pipe. This reliance on context-independency also makes pipelines relatively unsuited to symbolic processing, due to the highly context-dependent nature of such applications.

Pipelines usually incorporate fairly limited amounts of memory, at least with respect to the size of the images that they are used to process. By our earlier definition of "memory intensive" processing then, in which substantial amounts of memory are needed for each pixel of the image, pipelines are "not" memory-intensive. This is not an unequivocal criticism of pipelines though, since many operations involving the sorting of values within a local window (such as median filtering, for example) only require that enough memory be present to hold the kernel values that are sorted at any one moment.

Pipelines are thus quite effective for median filtering and similar operations.

In the area of object- versus coordinate-oriented processing, pipelines are generally better suited to the coordinate-oriented case. The performance of pipelines in object-oriented applications is a function of several factors. First, there is the question of the extent to which such processing is also context-dependent. It is usually quite so, leading one to think that pipelined machines would have great difficulty with it. On the other hand, the context-dependency in such cases in large part just results in alterations of the pattern of data access, rather than in any significant alteration of the instruction stream. Also, the processing being performed in such cases is frequently rather low-level, consisting only of binary decisions based on very simple sets of predicates (governing whether a particular pixel is or is not part of an object). This means that most object-oriented processing can be performed fairly early in a pipe, resulting in much shorter pipeline delays that would otherwise be encountered. Single pipelines are still limited by their single instruction stream to processing only one object at a time, but multiple pipes gain performance proportional to the number of pipes that may be effectively employed in any given application. Thus, pipelines as a general class rate slightly better than cellular machines for object-oriented processing.

11.9 MIMD ARCHITECTURES

MIMD stands for "Multiple Instruction, Multiple Data," and refers to architectures in which a number of largely autonomous processors are harnessed to process data in parallel. In this class of machines, each of the individual processors of the ensemble executes its own instruction stream. IU processing is typically performed in these systems by partitioning the image data among the processors in the array, with each element receiving a contiguous segment of the image.

Many MIMD architectures developed to date represent attempts to efficiently make use of the inexpensive, powerful, general- purpose microcomputer chips that are readily available in the commercial marketplace. The lure of such endeavors is obvious: In one sense, the "hard" part of the design of these chips has already been done; and, since the cost of this development has been amortized over perhaps millions of commercial chips, the design of the individual processors is virtually free to the system designer. Furthermore, many of these machines, particularly the more recent introductions to the field, are impressively powerful: Many of the newer chips surpass the capabilities of even "super minicomputers" of only a few years ago. If hundreds or even thousands of such general-purpose microcomputers could be efficiently connected into a processing network for IU applications, the reward would be significant. A good deal of work has already been done in this direction, and several research machines have already been built [51-55].

The drawback to all of this, of course, is to find a means of interconnecting and applying multiple general-purpose processors that is truly efficient. Synchronization and coordination overheads, resource contention, and programming considerations mean that the net increase in capabilities as more processors are added to the network is far from linearly proportional to the number of processors. Data flow within MIMD structures occurs in two main areas: between the local memories of the individual processors and

those processors themselves, and between the separate processors of the array. The link between processors and memories is no greater problem than in any conventional, "Von Neumann" machine. In fact, since the individual processors in an MIMD machine have a much smaller data base to deal with, the limitations of memory bandwidth are less apparent than in single-processor machines with larger, single memories. In all but the very lowest levels of IU processing though, information must be passed across the boundaries separating the image segments which have been assigned to the individual processors. This requirement is the source of most of the obstacles to achieving truly efficient operation of multiprocessor arrays.

There are a number of ways that processors can be interconnected in MIMD systems, each with its own advantages and liabilities. The various approaches can be categorized into three broad categories: Shared memory systems, bus-oriented systems, and "network" systems. In shared memory systems, the processors communicate with each other through overlapped regions in their memory spaces. That is, their memory systems are multi- ported, with regions of memory being accessible to more than one processor. Data is passed between processors by leaving "messages" in the shared memory areas [54]. Memory-based communication schemes have the advantage of incurring some of the lowest synchronization overhead costs for relatively local communications (communications between processors directly sharing memory space), but are much less efficient when the messages must cross more than one address-space boundary. In bus-oriented systems [52,55], a bus passes between processing elements, with messages being passed along the bus. In such systems, the transfer itself can be very rapid, regardless of the "distance" separating the communicating units, but the overall message bandwidth is limited to that available from the bus used. Finally, in "network" communication schemes, the individual processors are attached to nodes of a message-passing network of varying intelligence, which conducts message packets from the originating processors to their appropriate destinations [40]. Such networks have the advantages of low communications overhead for the individual processors, coupled with higher bandwidths than are available in a single-bus system. Since the network itself typically contains a fair amount of processing capability, and since the characteristics of such systems are sufficiently different from those of most other MIMD implementations, we will discuss their characteristics in greater detail in the subsequent section of this report entitled "Broadcast Architectures."

As we shall see in the following discussion of the performance characteristics of MIMD machines, their multiple instruction streams can be very useful for some types of processing. On the other hand, this greater flexibility carries with it a greater burden for the programmer, who must efficiently assign and coordinate tasks across the pooled computing resources. This programming overhead complicates the actual application of MIMD machines, and, coupled with the problems of interprocessor communication efficiency, has possibly contributed to their relatively slow rate of development and acceptance. As yet, the resource-allocation problem has remained too complicated and poorly understood to yield to effective handling by compiler software. This places further demands on the programmer.

11.9.1 Characteristics of MIMD Architectures

Due to their partitioning of the image between the multiple execution units composing their architecture, MIMD machines are best suited to processing involving only relatively local data access. "Global" algorithms, in which any local processor MAY be required to draw upon data from any location in storage are less efficiently handled by most MIMD machines.

In the area of computation-intensive versus memory-intensive processing, the relative performance of MIMD machines is largely a function of the characteristics of the individual processors composing the array, and the manner and degree to which the image data is partitioned between them. The ability of MIMD structures to perform computation-intensive processing is determined by both the number of processors in the array and by the execution speeds of the individual units. With current or near-future commercial microprocessors, performance levels in the 100-1000 MIP range are quite feasible. Ultimate capabilities are probably in the 10^4 to 10^5 MIP range. The upper limit is set both by the maximum likely VLSI uniprocessor performance figures, and by a maximum practical array size that is probably on the order of 10^3 to 10^4 processors. Larger arrays are rendered impractical by the limitations of communication networks and bus systems. The memory-intensive performance of MIMD machines is determined by the ratio between the size of the image segment assigned to each processor and the size of the local memories available. Obviously, the amount of per-pixel memory available is exactly this ratio. Most machines constructed to date have a fairly limited amount of per-pixel memory, primarily due to practical limitations such as funding level, intended scope of the project, etc. Increasing commercial memory density and uniprocessor memory addressing capabilities will probably result in increased MIMD performance in this area in the near future.

Since each local processor in an MIMD structure is a complete, general-purpose computer, MIMD machines are equally suited to both linear and non-linear, as well as both context- dependent and context-free processing. MIMD architectures as a class perform better in these areas than virtually any other architecture evaluated. On the other hand, if the region of data defining the context-dependency extends across the memory space boundaries of more than one processor, the problems of interprocessor communication and synchronization can have a substantial negative impact on both execution speed and programming efficiency.

The relative abilities of MIMD architectures to execute object-oriented and coordinate-oriented programs are rather similar to their just-discussed abilities to execute context-free and context-dependent programs. This is to be expected, since object-oriented processing is just a special case of context- dependent processing. The tradeoffs involved in designing a machine for efficient object-oriented processing once again revolve around the balance between local processor memory space and communications bandwidth. The single distinguishing feature of object-oriented processing is that there is no a priori information regarding the location of the data associated with any particular object being processed, other than a presupposition of connectivity. In MIMD structures, the most natural implementation of object-oriented algorithms is to assign a separate processor to each identifiable "object" in the image. Since any of these objects

may extend into any region of the image, it is necessary for each processor to have access to every part of the image. This access may be implemented either by supplying sufficient local memory for each local processor to store the entire image, or by providing a communications network memories for MIMD architectures. In general though, even in the present context of relatively limited processor memory spaces, MIMD architectures perform well on object-oriented problems, due to their multiple, independent instruction streams. Their coordinate-oriented processing performance, on the other hand, is somewhat less than that achieved by other machines having a greater inherent parallelism (eg: cellular machines with their tens or hundreds of thousands of arithmetic/logic units).

Finally, MIMD arrays are best suited to iconic processing, but show more promise than some for symbolic work as well. As is the case with other architectures though, machines which are optimized for symbolic processing are relatively poor for iconic processing and vice versa. A good deal of development has been done on the University of Mexico's "AHR" machine [71,72], which uses a bus-linked array of microprocessors for concurrent LISP processing. This machine shows good promise, but has the dual drawbacks of being a bus-structured machine, with the inherent communications bandwidth limitations that this implies, as well as relying on a sequential task-allocation unit that severely limits throughput in complex problems. Since this particular machine is really more of a data-driven architecture than it is an ordinary MIMD structure of the sort we have been discussing so far, we will defer further mention of it to the subsequent section on data-driven architectures.

11.10 NUMBER THEORETIC ARCHITECTURES

"Number Theoretic" architectures perform their arithmetic operations through the use of programmable look-up tables in fast RAM, employing residue arithmetic techniques to minimize look-up hardware requirements. They are characterized by very simple, highly regular circuitry, and extremely high throughput capacities. The single machine of this type constructed to date has exhibited throughputs of 250 million multiplications per second, and was built with 10 MHz NMOS circuitry [56].

Since residue arithmetic is rather unusual, we will present a brief description here of its operation. Readers wishing an exhaustive treatment of the subject are referred to the comprehensive volume authored by Tsabo and Tanaka [57].

In the early days of computing, look-up tables were sometimes employed for arithmetic processing when performance was paramount and cost no object. In this approach, every possible value of a function was stored in a large memory, ordered according to the values of the operands that would result in each output value. Arithmetic was then performed by using the values of the operands as addresses into this memory space, with the resulting output value simply being read at the memory's data output. Since the access time of such memories was significantly less than the computation time of the logic required to implement the functions in question, this approach resulted in arithmetic execution speeds orders of magnitude greater than were otherwise obtainable. The drawback to the technique was that very large memories were required for the look-up tables, even for relatively modest dynamic ranges. (A 64K x 16 bit table is required for the mul-

tiplication of two 8-bit operands.)

The residue number system provides a means by which the size of the required look-up tables may be significantly reduced for any given dynamic range. This reduction of the problem is accomplished through the application of elements of number theory [57]. The residue number system is based on a collection of N integers: M_1, M_2, ..., M_N; each of which is called a modulus. The moduli are required to be relatively prime, i.e. have no common factors, but need not be absolutely prime. The "residue" of X mod M_i is defined to be the least positive integer remainder of the division of X by M_i, where X is the number to be converted to residue form, and M_i is a modulus. The dynamic range of the full processor, D, is given by the product of the moduli employed, and any integer in the range of $0 < X < (D-1)$ can be uniquely represented. The residues of residue arithmetic may be thought of as somewhat analogous to the digits of conventional number systems.

The strength of the residue number system lies in the way in which arithmetic is performed within it. Arithmetic operations are performed in parallel, in each modulus, or base, and then combined to uniquely determine the final result. All operations are also performed in modular arithmetic, in which all carries are ignored, and only the remainders with respect to each base are retained. This further simplifies both the arithmetic operations and the hardware needed to implement them. An example of residue addition is given below. Here, we choose the moduli of 5, 7, and 8, giving a dynamic range D of 280. Thus, we may uniquely represent integers in the range of 0 to 279, without overflow. In the example, we add 19 to 87, to obtain a result of 106. We begin by converting 19 to residues of 4, 5, and 3, in the three moduli we have chosen. The 4 is the least remainder, or residue, of 19 mod 5; with the other residues shown similarly derived. We convert 87, the second operand in like fashion, and arrange the residues of the two operands in columns according to the moduli employed, as shown in Table I.

		$M_1(5)$	$M_2(7)$	$M_3(8)$
19	-->	4	5	3
+ 87	-->	+ 2	+ 3	+ 7
106	<--	1	1	2

TABLE I: Residue Arithmetic Example

Each column is then independently summed, with the results again expressed as residues of the associated moduli. The resulting number triplet (1,1,2) uniquely represents the result in the residue number system. That is, that the actual result gives a residue of 1 when reduced mod 5, a residue of 1 when reduced mod 7, and a residue of 2 when reduced mod 8. This result can be uniquely decoded to determine the integer result, 106, in the range of 0 to 279. One of two decoding processes may be used, known as the

mixed-radix and chinese remainder theorem methods. A full discussion of the details and relative merits of these decoding techniques is beyond the scope of this report, but may be found in references [56] and [57].

As the example shows, the arithmetic operations performed in each modulus are independent of each other, making them ideally suited to parallel computation. Furthermore, look-up tables can be used to perform the operations for each modulus, giving very high speed operation. Very large dynamic ranges may be obtained with relatively small look-up tables, by using a sufficient number of moduli. In effect, the residue number system permits the use of look-up tables which only increase in size as a logarithmic function of dynamic range, rather than as a linear function of dynamic range, as would be the case in a system using a conventional approach.

As mentioned above, residue systems exhibit a high hardware regularity, which makes them ideally suited to VLSI implementation. Also, since they merely involve the use of a special number system, they may be effectively combined with various architectural topologies to produce hybrid machines of even greater power, with little impact in the form of increased programming difficulty. As might be expected though, there are limitations to the use of residue number systems, one minor, and one severe. The minor limitation is that there is some amount of overhead associated with the conversion of data from a conventional representation into residue form and back again. The amount of overhead contributed by this requirement varies as a function of the amount of processing that can be performed in the residue domain. The conversion time is fixed at some small value, typically one memory cycle for encoding and N-1 memory cycles for decoding, where N is the number of moduli employed. Thus, residue arithmetic makes most sense in applications involving great numbers of operations that may be performed in the residue domain.

This brings us to the second, more serious limitation of residue number systems. Since residue arithmetic ignores all carries, it is impossible to perform magnitude comparisons within a residue representation. This means that any non-linear operations, such as thresholding, etc, necessarily involve the translation of the residue-represented numbers back into a conventional number system before the operation can be carried out This requirement does limit the range of application of residue arithmetic. The performance obtainable in the areas well suited to its use, however, make it a strong contender for many applications within IU.

11.10.1 Characteristics of Number Theoretic Architectures

Number theoretic techniques involve only the number system in which computations are performed and, as such, have little direct impact on the architecture of machines employing them. On the other hand, the unavailability of magnitude comparisons in in the residue representation, and the overhead involved in converting between residue and normal representations dictate the use of residue techniques in applications dominated by computational requirements and characterized by a lack of both non-linearity and context dependency. The single machine constructed to date using a residue number system was optimized for convolution operations on 5 x 5 kernels, a local, linear, computation-

intensive, coordinate-oriented, iconic, and context- free operation.

Number theoretic machines may be designed that are well- suited for either local or global operations. However, the extremely high throughputs available when using residue arithmetic dictate that image memory access be highly optimized, in order keep up with the processing circuitry. This probably means that most practical residue-based machines will be optimized for local processing.

As just mentioned, residue-arithmetic based systems cannot perform any sort of magnitude comparison or thresholding, and therefore are not at all suited to either non-linear or context- dependent processing. This restriction must be qualified by pointing out that, while residue arithmetic can't handle these types of operations, systems may be built which incorporate both residue-based processing circuitry and conventional arithmetic elements, thus providing the ability to perform both non-linear and context-dependent processing. The fact nonetheless remains that heavily context-dependent processing, such as that found in object-oriented applications, will benefit but little from the availability of residue hardware. This is due to the fact that most of the processing in such applications consists of the sort of decision-making that is impossible with residue techniques.

Finally, since residue arithmetic is a number system, architectures based on its use are only suitable for use in iconic processing: There is no foreseeable use for residue techniques for strictly symbolic processing.

11.11 SYSTOLIC ARCHITECTURES

Systolic architectures are best described as a sub-class of pipelined architectures. They are similar to more conventional pipelines in that they may be conceptualized as "numeric assembly lines," with each element in the array accepting data from its predecessors and passing its results to its successors in the network. Systolic structures are different, however, in that they employ arrays of identical function modules called cells; in that these arrays are most often two dimensional; and in that data flow through the cells of the systolic array is highly regular. They were developed by, and have been popularized in large part by H. T. Kung and his students [58-61], [62], and are somewhat similar to residue-based systems in that they seem to be well-suited to highly computation-intensive applications, although they lack characteristics which would permit broader usage.

Systolic processors operate by exploiting data and processing regularities within the algorithms being implemented. Their design principle is to achieve the highest level of parallel-pipelined processing possible within an algorithm, while making maximally efficient use of data fetched from memory. This is typically done by operating on serial data streams fed into a multiply-connected array of identical processing cells. In such systems, the input data streams usually flow into the array from one or more separate directions, with computations being performed whenever appropriate elements of the criss-crossing data streams "meet" each other in one of the processing cells.

Systolic arrays have a number of advantages which recommend their use for image understanding systems. One of the most important characteristics of systolic hardware is its extreme regularity, which combines with its relatively simple communication requirements to make such systems almost ideal for VLSI implementation. Also, as mentioned above, the pipelined nature of systolic architectures does much to alleviate the memory bandwidth problems encountered in other architectures. Furthermore, in purely local operations, the size of the array is dictated purely by the size of the kernel being processed, independent of the size of the image being manipulated. (Systolic structures share this characteristic with number theoretic processors.)

The preceding advantages present strong arguments for using systolic methods to meet many of the requirements of IU processing. Unfortunately, the realm of application of systolic hardware is limited to many of the same areas also served by the number theoretic architectures discussed in the last section. These limitations arise from the fact that systolic arrays rely heavily on certain fixed sequences of operations and regularities of data access in the algorithms for which they are designed. As such, they represent optimized solutions for specific applications of necessarily limited generality. If an application does not meet the timing and functional constraints for which a systolic array was designed, it may not be possible to implement it on that machine This means that systolic arrays may be very useful in dedicated imaging systems of the sort anticipated for future intelligent military systems, but that they will have only limited usefulness in a research environment.

11.11.1 Characteristics of Systolic Architectures

As we have already mentioned, systolic machines are most suited to local processing. Because they operate on serial streams of data, the inherent "pipeline delay," and the size of the required array will be strong functions of the linear extent of the kernels being processed. This would make systolic machines poor candidates for applications in which local results depended strongly on global data values.

To their credit, systolic machines have no difficulty with non-linear processing, as did the number theoretic machines of the preceding section. This is to be expected, since they do not rely on special number systems for their speed. On the other hand, due to their rigid functional and timing restrictions, they also perform poorly in either context-dependent or object- oriented applications. Likewise, since the size of a systolic array seldom matches that of the image being processed, they are generally less suited for memory-intensive applications than they are for purely computation-intensive ones. Finally, systolic machines are exclusively limited, at least as of this writing, to processing in the iconic domain.

11.12 HIERARCHICAL ARCHITECTURES

Hierarchical architectures encompass both data structures and particular hardware configurations. Known by various names, including "pyramids," "cones," "structured vision systems," "hierarchical systems," and "parallel-serial architectures;" such systems are attempts to model the manner of functioning of natural (human and animal) visual systems

[63]. They employ a system of multiple levels of resolution, as do the organic systems they mimic. This approach has some valuable consequences for image understanding applications, as we will see in the following "characteristics" sub-section.

Hardware for hierarchical processing would have log 2(N) separate arrays of identical execution units, where N is the linear dimension of the N x N pixel input image. The lowest level of the architecture has a full N x N array of processing elements, with each succeedingly higher level having 1/2 the linear dimension and resolution, meaning 1/4 the number of processing elements. The highest level of the structure consists of but a single processor. Prior to processing, an image is loaded into the lowest level of this "pyramid," and data values for each higher layer are computed by taking the average of the four cells associated with each processor, on the level below. During the processing of an image, data is passed back and forth, both between cells of the same level (primarily on a nearest- neighbor basis), as well as between cells of different levels. Structures proposed to date typically require connection of each processor with four adjacent cells on its own level, as well as with four cells on the level above it, an implementations have been proposed, no actual machines embodying these concepts have been built, at least as of this writing. One such system is presently under construction by Boeing though, and detailed reports of its performance should soon be forthcoming [67]. Possible reasons for the lack of actual implementations of pyramid structures is the complexity, and the sheer volume of the hardware required. This last may not be as great a restriction as one might imagine though, since a hierarchically structured processor of any given resolution would contain only a third as many elements as a simple cellular machine of only twice greater resolution. For example, consider that a 64 x 64 pyramidal machine would contain only 5461 processing elements, as compared to the 16,384 elements of a 128 x 128 cellular machine, such as the Goodyear MPP. Given their advantages for various types of image understanding processing, which we will discuss below, it is likely that more such machines will appear in the near future.

11.12.1 Characteristics of Hierarchical Architectures

Since they are little more than a stack of cellular machines of successively decreasing resolution, hierarchical architectures exhibit many of the desirable characteristics of cellular ones. As with cellular machines, their extreme parallelism permits the use of serial arithmetic in the individual processing elements, while still achieving very high throughputs. The simplicity and compactness of serial circuitry permits very high cell densities, as well as high circuit yields in the individual cells. Also, the high regularity of the array hardware and the interprocessor communication links makes such designs natural candidates for VLSI implementation through the use of CAD techniques.

Hierarchical architectures also share the excellent performance of cellular machines in applications involving local and linear processing. Like cellular machines too, their performance at global and non-linear processing is very good, with the specific levels attainable being largely dependent on the implementation. Versions with more flexible communications networks and greater amounts of memory dedicated to mask vector storage will perform proportionately better in these areas than will machines without such resources.

The memory available in hierarchical systems is subject to the same tradeoff against processing circuitry and, generally, array size as we observed in the case of cellular machines. The more memory allocated to each processing cell, the more area each will require, and the more silicon real estate will be needed by the system as a whole. On the other hand, since array machines are usually constructed with linear array dimensions expressible as integral powers of two, and since hierarchical machines will for the most part have arrays that are a factor of two smaller than cellular machines of equivalent technology level, there will no doubt be greater amounts of local memory in most hierarchical machines than in equivalent cellular ones. Hence, hierarchical machines should be slightly better than cellular machines for memory-intensive applications, but somewhat slower in computation-intensive ones.

Since the hierarchical machines proposed to date are all SIMD machines, they have the single-instruction-stream limitation of their cellular predecessors. This contributes to a context- dependent performance that is somewhat less than that attainable on equivalent, context-free processing. It should be pointed out though, that the single instruction stream is by no means a requirement of the hierarchical data structure on which these machines are based. Furthermore, the overlaying of multiple resolution levels in the data structure can greatly facilitate many types of context-dependent processing, by providing a "lateral lookahead" capability. This can be especially useful in object-oriented processing and iconic to symbolic translation. In object-oriented processing, hierarchical machines can make use of the implicit locality of data associated with a particular object. "Implicit locality" refers to the fact that, while the data associated with an object cannot a priori be assumed to occupy any given area of the image, it is likely that an object's data will be found in some fairly localized region. This means that information regarding the extent of an object can readily be passed through the multiple-resolution data structure to guide the operation of the individual processors at each level of resolution. For this reason, hierarchical architectures seem to hold excellent potential for dealing with the difficult iconic- symbolic translation problem.

While no work has been done to date on the application of hierarchical architectures of the sort discussed here to symbolic processing, their structure closely matches that of so-called "tree machines," which have been expressly designed to execute symbolic algorithms. Thus, they should be easily adaptable to use in symbolic applications.

11.13 DATA-DRIVEN ARCHITECTURES

Data-driven, or so-called "data-flow" architectures represent an attempt to "automatically" exploit the parallelism inherent in any given algorithm. This is done through the use of hardware structures in which the execution of each individual arithmetic or logical operation is triggered by the availability of the data forming its operands. Thus, the lowest level of data is presented to the lowest level of the architecture, and is routed to computation units embodying the various functions of the program being executed. This data-routing is accomplished by the hardware, which previously has been configured to implement the application program. Each of the lowest-level computation units receives the operands it needs to perform its programmed function. As each unit completes its function, it passes the results on to subsequent units which await those results for use as operands to their own programmed functions. Whenever a functional

unit has received all of its required operands, it "fires," and passes its results in turn to other units awaiting them. In this fashion, data ripples through the ensemble of computational elements, proceeding as rapidly and with as much concurrency as is permitted by the algorithm being implemented.

- Hardware ensembles for data-driven processing would ideally have as many execution units as there were independent functional blocks within the application program being implemented. Some proposed architectures do, in fact, recommend this approach [68- 70]. Others, such as the AHR computer being developed at UNAM [71,72], use instead a multitude of general-purpose execution units, each of which can perform a number of possible operations. In such machines, the availability of operands, and the nature of the functions with which they are associated, are kept track of by some central dispatching unit. This central dispatcher (called the "grill" in the AHR machine) determines when a function is ready to "fire," assigning execution units from a pool of available units as required.

Both approaches just mentioned have relative strengths and weaknesses. The first approach, in which there is a unique functional element for every functional block in the program being implemented, provides the greatest possible execution speed for any given program. A drawback of this approach is that it requires an enormous amount of processing circuitry, a great deal of which will lie idle during most of the course of program execution. Also, reconfiguration of the processing ensemble for programs having radically different directed graphs can be difficult. Machines of this type tend to be best suited to small classes of substantially similar applications: Radically different algorithms require different system architectures. Systems relying on a central "dispatching engine" largely eliminate this problem by providing a more flexible, general- purpose structure that is readily adapted to the varying demands of different algorithms. This flexibility is achieved at some cost in absolute performance, however. By concentrating the dispatching and control functions in a single, central processing unit, this approach performs those tasks serially, potentially creating a bottleneck slowing the overall performance of the architecture.

All data-driven architectures share the problem of complexity control. This term refers to the efficient decomposition of algorithms into functional blocks to be distributed to the individual execution units of these machines. The most simple-minded approach is to divide the algorithm into its simplest discernable component operations, such as addition, multiplication, comparisons, etc. The problem with this approach is that it requires an enormous number of execution units for all but the most trivial of problems. As the number of execution units required grows, so does the number and complexity of interconnections between them. On the other hand, as the level of processing assigned to the individual units is raised, opportunities for parallel execution are lost. The optimal tradeoff between these two factors varies considerably from application to application, and the means of arriving at that tradeoff are poorly understood at best. The lack of understanding of this crucial area limits the current and near- term utility of data-driven architectures.

11.13.1 Characteristics of Data-Driven Architectures

The performance characteristics of data-driven architectures are highly dependent on the nature of the particular hardware embodiment involved. As such, it is difficult to draw conclusions regarding the performance characteristics of these machines that will be true for all such architectures. A few broad observations can be made, however.

In general, data-driven machines will perform context-free algorithms with greater efficiency, if not with greater speed, than they will algorithms that are context-dependent. This because they require that each potential branch of a program be expressed in the form of additional blocks of execution units. In the majority of algorithms, only one such branch will be executed at any one time. That is, most branches are of the "either-or" form. This means that much of the hardware associated with the branches not taken will lie dormant much of the time. Thus, while context-dependent programs will execute with as much parallelism as is permitted by the algorithm, data- driven machines built for their execution will exhibit substantial inefficiencies in their usage of hardware.

For similar reasons, it can be said that data-driven machines are better suited to computation-intensive, rather than memory intensive algorithms. Because each execution unit must wait for all of its various operands to appear before it can "fire," data-driven structures will be most efficient in applications in which the ratio of execution units to operands is high. In other words, the fewer operands an execution unit requires, the more frequently it will be able to fire. In memory-intensive algorithms, there would logically be fewer execution units operating at any one time, since more intermediate values would be generated for each image pixel, before the next stage of processing could begin. As a corollary to this, data-driven machines would be largely unsuited for use in object-oriented applications, due to the extreme data- dependency involved in such algorithms.

Finally, data-driven architectures, at least as represented by the AHR machine, seem to hold some potential for use in symbolic applications. The realization of this potential will become more fully evident as the problem of bottlenecking in the task distribution hardware are addressed and overcome in the future.

11.14 "BROADCAST" ARCHITECTURES

"Broadcast" architectures are constructed from arrays of processor/memory cells, which appear as the nodes of a locally- connected communications network. The processors at each cell are very simple, consisting of only a few registers, a rudimentary ALU, and a rule table that is typically shared with other, adjacent processors. Each cell also contains a communications processor which is interfaced to and forms part of a packet-switched communications network. All communications between cells occurs through this packet-switched network, in the form of discrete "messages" passed between members of the array. In normal operation, each cell knows the addresses of those with which it must communicate. A message is formed by attaching a destination address to the block of data that is to be transferred. This packet is then released into the network, where it is accepted and passed on by the various communications nodes associated with each of the cells of the array. The circuitry at each node routes the

message to one of its neighbors, depending on which of those neighbors is closest to the ultimate destination of the packet. When the message reaches its destination, it is removed from the network, and passed to the local processor for which it was intended.

In typical applications, each processing element of the array "knows" the addresses of several other cells with which it communicates. Unconnected cells may establish communication links through the use of "message waves." A cell that wishes to search for another with particular characteristics broadcasts a message wave through the network, indicating the type of cell that is being searched for, and the address of the originating cell. When a cell matching the specified characteristics is found, it transmits its address back to the requesting cell, indicating its availability. The requesting cell responds to the first such message with an "accept" message, and to all subsequent ones with "reject" messages. Once a cell of the desired type has been found, the requesting cell also broadcasts a special, high-speed "cancel" message wave, which overtakes and cancels the original search wave.

Broadcast architectures were developed at MIT by Hillis and coworkers [40] to provide for the concurrent processing of semantic nets. Advantages claimed for such processors (the proposed MIT machine is called the "connection machine") are that: 1) They are able to implementa all of the operations found in relational algebra, as well as structured inheritance networks. 2) The individual processors are very simple, and well-suited to VLSI implementation. 3) The communication network, being locally connected, involves only short wires, and packs easily into two dimensions. 4) Because all connections between cells are virtual ones, made solely through the packet- switched communications network, their structure may readily be altered to mimic the structure of any specific problem. 5) Because all links between objects are made in software, the physical location of the objects themselves can be changed easily, as long as the (relatively few) cells communicating with the displaced objects are informed of their changed location. This could have valuable implications for issues such as fault tolerance and virtual storage in these systems.

11.14.1 Characteristics of Broadcast Architectures

Our discussion of the characteristics of broadcast architectures should be prefaced with the observation that they were designed with the primary goal of providing concurrent execution for symbolic algorithms. As such, they are poorly suited to most forms of iconic processing. Since the bulk of our software metric deals with iconic problems, the broadcast machines fare rather poorly in comparison with others developed primarily to meet iconic processing requirements. On the other hand, this poor performance in iconic applications is almost exclusively a result of the characteristics of the packet- switched communications scheme, particularly in the case of messages which need only travel short distances.

Several factors affect the efficiency of the packet-switched network. First, is the overhead associated with assembling data and destination addresses into message packets. This process obviously takes time that is not required in, for instance, a cellular architecture, where local communications are globally controlled. In many iconic applica-

tions, where the flexibility of the packet-switched approach is not needed, this overhead can present a severe liability. The second efficiency-limiting factor is the response of the packet-switched network to varying message loads. The distributed routing approach, in which individual communications nodes independently decide how to route outgoing messages, depends on relatively light traffic loads to avoid delays and potential deadlocks as messages collide. Finally, since most large arrays of this sort employ serial data channels between processors, the explicit address information passed along with the messages will reduce the overall bandwidth available for data.

These limitations of broadcast architectures for iconic processing could easily be overcome though, if an alternative provision were made to permit nearest-neighbor communication without the mediation of the packet-switching hardware. This would allow the array to operate as a conventional cellular machine, with the concomitant advantages for iconic processing exhibited by that architecture.

Given that they weren't designed for iconic processing, we would be better able to perform context-dependent processing than other types without such local program storage.

Broadcast architectures could potentially perform well in object-oriented processing, due to their local program storage, and the independent message-passing capabilities of the unit processors. Care would have to be taken in the actual implementation of the algorithms, however, to ensure that the local bandwidth capacity of the message network is not overloaded by the message-passing requirements of the application. Specific implementations of object-oriented metric algorithms would have to be executed to accurately determine the performance of these machines in this area.

11.15 ASSOCIATIVE ARCHITECTURES

Associative architectures are those in which data is addressed by value or structure, rather than by absolute memory location. A number of machines based on this storage concept have been built or proposed [73-78], with the STARAN machine representing the highest degree of sophistication of that field.

There are two primary means of providing a memory system with associative capability. The most direct method is to assemble the memory from content-addressable cells Memories of this sort produce a "data present" signal in response to a word presented at their data inputs if they contain a piece of data matching that being inquired about. This is in contrast to a conventional memory which provides a piece of data in response to an address or location-indicating word. Alternatively, an associative memory may be constructed from arrays of conventional memory cells, address-sequencing counters, and comparator circuitry. Systems of this sort mimic the function of true content-addressable systems by loading the word being scanned for into a comparison register, and then quickly sequencing through all words present to see if any produce a match with the reference data. A flag output is set if a match is found, and a "no match" indication is given if the search did not succeed. The true content-addressable approach is by far the fastest of the two methods, but in most cases requires substantially more

hardware to implement.

The associative nature of these architectures though, comes more from the organization of data within their memory systems than it does from the cell-level addressing method that is used to implement them. Associative machines are usually characterized by very large word lengths in their memories. Related pieces of data are stored in various fields within single words. In this fashion, once one has located one piece of data corresponding to a particular object, the location of other information regarding that object is immediately known, since it is located in the same physical word of storage. This partitioning of information within single memory words is usually facilitated by hardware allowing the scanning, reading and writing of selected word fields, without disturbing the information stored in other fields of the same words.

Associative capability of the sort discussed here finds greatest use in applications involving searches of large databases for information structures with particular attributes Examples of such use might be the querying of image databases for objects having certain characteristics, or the following of multiple radar tracks in a complex air-defense system. Associative machines also appear to be very well suited to the requirements of knowledge-based systems of the sort encountered in the field of artificial intelligence This is because a great deal of the processing load imposed by such systems arises from searching for "rules" that are applied in response to a set of conditions in the database. By performing searches across the rule base in parallel, associative machines promise far greater throughputs than conventional serial machines in these applications.

11.15.1 Characteristics of Associative Architectures

The characteristics of associative architectures revolve around their memory access methods, and even under the broad classification of "associative," a wide variety of different memory structures may be found. The nature of these memory structures will have a great deal to do with the suitability of the machines involved for different sorts of processing. A few general observations may be made, however.

In most areas of iconically-based processing, associative machines as a class are undistinguished by either very good or very bad performance when compared with the other types of processors that we have discussed earlier. Most machines actually constructed to date suffer somewhat in overall performance when compared to, for instance, cellular numeric arrays in the areas of local, linear, context-free processing. This, however, is more a reflection of technological limitations that were in force at the time that these machines were constructed tan it is indicative of any inherent deficiency in the class as a whole. These limitations dictated that fairly small, memory-intensive processing ensembles be built, in the interests of practicality. More modern VLSI-based architectures, by taking advantage of the extreme circuit densities made possible by current technology, promise to largely overcome this restriction.

The real strength of associative architectures appears to be in their ability to rapidly perform the pattern-match searches that are so integral to symbolic processing. Some tasks in symbolic processing are inherently serial, but a great deal of the computational

load found in most knowledge-based systems involves searching and matching functions. Associative machines seem to hold great promise in this area.

11.16 SUMMARY AND CONCLUSIONS

In this study, we have comprehensively examined the processing requirements of image understanding, and have studied various computer architectures intended to satisfy them. It was concluded that the best means of dealing with the proliferation of IU algorithms was to extract from them the highest level of operations that were common to a wide range of algorithms in various areas. A software taxonomy was developed, based on the demands various algorithms placed on the hardware which executes them. This taxonomy was used to determine the different classes of computation represented by IU algorithms, and subsequently to select representative algorithms to serve as a hardware metric set. Specific suggestions were made regarding the elements of this metric set, which can be used to objectively evaluate the performance of various concurrent computing structures with radically different architectures.

Next, we turned to the problem of evaluating generic classes of processor architectures, again basing the evaluation on the software taxonomy presented in the first section of this report. A total of nine architectural families were studied, with specific comments made regarding each.

The overall objective of this report has been to develop a knowledge base that would facilitate the inference of broadly based conclusions regarding the suitability of various computer architectures for image understanding. The conclusions thus drawn are as follows:

First, processor designs optimized for iconically- represented problems enjoy a much greater degree of evolution and sophistication than do those for symbolic problems. This is probably because iconic image processing is necessary to extract the symbolically structured information that is the subject of symbolic processing. Early researchers in the field were most immediately impressed with the vast throughput required by so- called "low-level," iconic processing, and so concentrated the bulk of their efforts there. Only in recent years has the realization developed, chiefly among workers in the artificial intelligence (AI) community, that symbolic problems can involve far greater computational requirements than had previously been supposed. Indeed, it now appears that many problems in symbolic computation can only be solved in exponential time, rather than the polynomial time characteristic of many iconic problems. Thus, one conclusion of our study is that special architectures for symbolic computation are still in their infancy, and therefore presently represent the weakest link in the effort to develop autonomous vision-based systems.

A second conclusion of the study is that there is another class of processing, logically separate from either iconic or symbolic, because it contains elements of both, intimately interwoven. This is the area of iconic-to-symbolic translation. Few existing or proposed architectures adequately meet the requirements of this process. The difficulty posed by iconic-to- symbolic translation is that it is a fundamentally object- oriented

process, with few a priori restrictions imposed on the extent of data associated with any particular object being processed. Single instruction stream machines perform poorly in this area, because they can only process data associated with a single object at a time. Multiple instruction stream machines, on the other hand, must solve the problems of efficient data partitioning and interprocessor communication and coordination. The architectures which show most promise for solving these problems are those belonging to the class of "hierarchical," or "pyramid" machines. These architectures employ multiple, overlaid levels of resolution in their representation of image data. These multiple resolution levels can be used to exploit the connectivity that is implicit in object-oriented processing.

Third, again in the area of purely symbolic processing, a great deal of the computational load of such applications involves searching data structures to find matches or common elements. Since this search capability is an inherent characteristic of associative machines, they would appear to be natural candidates for certain areas of symbolic computation.

Assembling these conclusions into a final, comprehensive recommendation for the "ultimate IU processor," it appears that the most useful piece of hardware to have, for research, strategic, and tactical application, would be some sort of "Image Database Engine." This machine would accept raw image data as input, convert it to symbolic form, and then serve as an intelligent symbolic database manipulation processor, allowing a more conventional processor to rapidly query the symbolic database developed from the input image data. A possible form that such a processor might assume is even fairly easy to see, given our earlier examination of the various concurrent processor classes. Hierarchical structures would appear to be well-suited to this sort of application, due to their excellent capabilities in the realm of iconic processing, as well as their potential for iconic-to-symbolic translation. Furthermore, the addition of an associative capability to the basic pyramidal structure would provide the ability to serve as the sort of intelligent symbolic query processor, once the image information had been rendered in that form. Thus, a single machine would be able to perform all levels of iconic processing; translate the iconic information into symbolic form; and finally, to handle the most time- consuming portions of symbolic processing. It is possible that the such a machine could also be capable of executing all levels and forms of symbolic processing, but a definitive determination of this would require further study.

REFERENCES

[1] Proc. 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Hot Springs, Virginia, Nov 11-13, 1981, IEEE Press, New York, 1981.

[2] Fu, K.S. and Ichikawa, T., "Special Computer Architectures for Pattern Processing," CRC Press, Florida, 1982.

[3] Duff, M.J.B., and Levialdi, S., "Languages and Architectures for Image Processing," Academic Press, London, 1981.

[4] H.T. Kung, ed., "VLSI Systems and Computations," Computer Science Press, 1981.

[5] Proceedings of the Fifth International Conference on Pattern Recognition, Dec 1-4, 1980, Miami Beach, IEEE Computer Society Press, New York, 1981.

[6] Proceedings of the Sixth International Conference on Pattern Recognition, Oct 19-22, 1982, Munich, Germany, IEEE Computer Society Press, New York, 1982.

[7] Fu, K.S., "Special computer architectures for pattern recognition and image processing -- An overview," AFIPS conference proceedings, 1978, pp. 1003-1013.

[8] Rosenfeld, A., and Kak, A., Digital Picture Processing, 2nd edition, Academic Press, New York, N.Y., 1982.

[9] Japan Information Processing Development Center, Proc. Int'l Conf. Fifth Generation Computer Systems, Tokyo, Oct 19-22, 1981.

[10] Danielsson, P.E., and Levialdi, S., "Computer Architectures for Pictorial Information Systems," Computer, Vol. 14, No. 11, Nov. 1981, pp. 53-67.

[11] Hwang, K., and Briggs, F.A., Computer Architectures for Parallel Processing, McGraw-Hill, New York, N.Y. (in press).

[12] Onoe, M., Preston, K., and Rosenfeld, A., eds., Real- Time/Parallel Computing: Image Analysis, Plenum Press, New York, N.Y., 1981.

[13] Preston, K., and Uhr, L., eds., Multicomputers and Image Processing, Academic Press, New York, N.Y., 1982.

[14] Uhr, L., Computer Arrays and Networks: Algorithm-Structured Parallel Architectures, Academic Press, New York, N.Y., 1982.

[15] Preston, K., "Comparison of Parallel Processing Machines: A Proposal," in [3], pp. 305-319.

[16] Preston, K., "Image Manipulative Languages: A preliminary Survey," in Pattern Recognition in Practice, Kanal, L.N. and Gelsema, E.S., eds., North-Holland, Amsterdam, 1980.

[17] Swain, P.H., Siegal, H.J., and El-Achkar, J., "Multiprocessor Implementation of Image Pattern Recognition: A General Approach," in [5], pp. 309-317.

[18] L.S. Davis, "A Survey of Edge Detection Techniques," Computer Graphics and Image Processing, Vol. 4, No. 3, September 1975, pp. 248-270.

[19] R.A. Kirsch, "computer Determination of the Constituent Structure of Biological Images," Computers and Biomedical Research, Vol. 3, No. 3, June 1971, pp. 315-328.

[20] R. Nevatia and K.R. Babu, "Linear Feature Extraction and Description," Computer Graphics and Image Processing, Vol. 13, 1980, pp. 257-269.

[21] C.R. Brice and C.L. Fennema, "Scene Analysis Using Regions," Artificial Intelligence, Vol. 1, Fall 1970, pp. 205-226.

[22] S.W. Zucker, "Region Growing: Childhood and Adolescence," Computer Graphics and Image Processing, Vol. 5, 1976, pp. 382-399.

[23] E.M Riseman and M.A. Arbib, "Computational Techniques in the Visual Segmentation of Static Scenes," Computer Graphics and Image Processing, Vol. 6, No. 3, June 1977, pp. 221-276.

[24] A. Rosenfeld and M. Thurston, "Edge and Curve Detection for Visual Scene Analysis," IEEE Transactions on Computers, VI. C-20, May 1971, pp. 562-569.

[25] D. Marr, "Early Processing of Visual Information," Philosophical Transactions of the Royal Society of London, B275, 1976, pp. 483-524.

[26] W.K. *Pratt, Digital Image Processing, pp.* 330-333, John Wiley & Sons, New York, N.Y., 1978.

[27] R. Ohlander, "Analysis of Natural Scenes," Computer Science Department Report (Ph.D. thesis), Carnegie-Mellon University, Pittsburgh, 1975.

[28] R. Ohlander, K. Price, and D. Raj Reddy, "Picture Segmentation Using a Recursive Region Splitting Method," Computer Graphics and Image Processing, 1978, pp. 313-333.

[29] S. Tsuji and F. Tomita, "A Structural Analyzer for a Class of Textures," Computer Graphics and Image Processing, Vol. 2, 1973, pp. 216-231.

[30] J. Grinberg, G.R. Nudd, and R.D. Etchells, "A Cellular VLSI Architecture for Image Analysis and Two-Dimensional Signal Processing," (forthcoming in IEEE Computer, 1983).

[31] M.J. Hannah, Computer Matching of Areas in Stereo Images, Stanford AI Laboratory Memo AIM-239, July, 1974 (Ph.D. thesis)

[32] D.B. Gennery, "Modelling the Environment of an Exploring Vehicle by Means of Stereo Vision," Stanford AI Laboratory Memo AIM 339, June 1980.

[33] H.H. Baker and T.O. Binford, "Depth from Edge and Intensity Based Stereo," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver, Canada, August, 1981, pp. 631-636.

[34] R.D. Arnold, "Local Context in Matching Edges for Stereo Vision," Proceedings of Image Understanding Workshop, Cambridge, Mass., May 1978, pp. 65-72.

[35] R. O'Gorman and M.B. Clowes, "Finding Picture Edges Through Collinearity of Feature Points," Proceedings of the Third International Joint Conference on Artificial Intelligence, August 1973, Stanford, Calif., pp. 543-555.

[36] A. Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection," Communications of the ACM, February 1976, pp. 73-83.

[37] H. Blum, "A Transformation for Extractiong New Descriptions of Shape," in Symposium on Models for Perception of Speech and Visual Form, W. Wathen-Dunn (ed.), MIT Press, Cambridge, Mass., 1967, pp. 362-380.

[38] T.O. Binford, "Visual Perception by a Computer," IEEE Conference on Systems and Controls, Miami, Florida, December 1971.

[39] R. Nevatia and T.O. Binford, "Description and Recognition of Curved Objects," Artificial Intelligence, Vol. 8, No. 1, February 1977, pp. 77-98.

[40] Hillis, W.D., "The Connection Machine (Computer Architecture for the New Wave)," MIT A.I. Memo #646, September, 1981.

[41] Batcher, K.E. "Design of a Massively Parallel Processor;" Trans. IEEE on Comp., C-29, pp. 836-840, 1980.

[42] Duff, M.J.B., "Review of the Clip Image Processing System," Proceedings of National Computer Conference, pp.1055-1060, 1978.

[43] Hunt, D.J., "The ICL DAP and its Application to Image Processing," in [3], pp.275-282.

[44] R. D. Etchells, J. Grinberg, and G.R. Nudd, "The Development of a Three-Dimensional Circuit Integration Technology and Computer Architecture" Society of Photographic and Instrumentation Engineers, Vol. 282, pp. 64-72.

[45] Reeves, A.P., "A Systematically Designed Binary Array Processor," Trans. IEEE on Comp., C-29, pp. 278-287, 1980.

[46] Technical Staff, Hughes Research Laboratories, Computer Architectures Section, "The Associative Square Array Processor (ASAP)," Hughes white paper submitted to Cmdr. R. Ohlander, DARPA IPTO, June, 1983.

[47] F.A. Gerritsen and L.G. Aardema, "Design and Use of DIP-1: A Fast, Flexible, and Dynamically Microprogrammable Pipelined Image Processor," Pattern Recognition, Vol. 14, pp. 319-330.

[48] Granlund, G.H., "The GOP Parallel Image Processor," in Digital Image Processing Systems (eds. Bolc and Kulpa), Springer-Verlag, Berlin, pp. 201-227, 1981.

[49] R.M. Lougheed, et al., "Cytocomputer architectures for parallel image processing," Proc. IEEE Workshop on Picture Data Description and Management, Asilomar, USA, 1980, pp. 281-286.

[50] S.L. Tanimoto and J.J. Pfeiffer, Jr., "An Image Processor Based on an Array of Pipelines," Proc. 1981 IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management, pp. 201-208.

[51] Kruse, B., Danielsson, P.E., and Gudmundsson, B., "From PICAP I to PICAP II," in [2], pp. 127-156.

[52] Rieger, C., "ZMOB: Doing it in Parallel," in [1], pp. 119- 124.

[53] J.T. Kuehn and H.J. Siegel, "Simulation Studies of PASM in SIMD mode," in [1], pp. 43-50.

[54] W. Wulf and C. Bell, "C.mmp -- a jultiminiprocessor," Proc. 1972 Fall Joint Computer Conf., Dec. 1972, pp. 765-777.

[55] R. Swan, S. Fuller, and D. Diewiored, "Cm*: a modular, multimicroprocessor," AFIPS 1977 National Computer Conference, June 1977, pp. 637-644.

[56] Fouse, S.D., Nudd, G.R., and Cumming, A.D., "A VLSI Architecture for Pattern Recognition Using Residue Arithmetic"

[57] Szabo, N., and Tantaka, R., Residue Arithmetic and its Applications to Computer Technology, McGraw-Hill, New York, NY, 1967.

[58] H.T. Kung and S.W. Song, "A systolic 2-D convolution chip," in Non-Conventional Computers and Image Processing: Algorithms and Programs, Leonard Uhr, ed., Academic Press, 1981.

[59] H.T. Kung, "Let's Design Algorithms for VLSI Systems," Proceedings of Conference on Very Large Scale Integration: Architecture, Design, Fabrication, Caltech, January 1979, pp. 65-90.

[60] H.T. Kung and R.L. Picard, "Hardware Pipelines for Multi- Dimensional Convolution and Resampling," in [1], pp. 273- 283.

[61] Blackmer, J., Frank, G., and Kuekes, P., "A 200 Million Operations per Second (MOPS) Systolic Processor," Proceedings of SPIE Real-Time Signal Processing IV, pp. 10- 18, August 25-28, 1981, San Diego.

[62] Nash, J.G., Hansen, S., and Nudd, G.R., "VLSI Processor Arrays for Matrix Manipulation," in VLSI Systems and Computations, pp. 367-378, ed. H.T. Kung, Computer Science Press, 1981.

[63] S. Tanimoto and A. Klinger, eds., Structured Computer Vision, Academic Press, 1980.

[64] A. Rosenfeld and L.S. Davis, "Hierarchical Relaxation," in Computer Vision Systems, A.R. Hanson and E.M. Riseman, eds., Academic Press, New York, N.Y., 1980.

[65] S.L. Tanimoto, "Regular hierarchical image and processing structures in machine vision," in Computer Vision Systems, Hanson and Riseman eds., Academic Press, New York, N.Y 1978, pp. 165-174

[66] L. Uhr, "Layered "recognition cone" networks that preprocess, classify, and describe," IEEE Trans. Computers, Vol. 21, pp. 758-768.

[67] S.L. Tanimoto, Personal communication to the authors, May, 1983.

[68] J.B. Dennis and D.P. Misunas, "A Computer Architecture for Highly Parallel Signal Processing," Proceedings of the ACM 1974 National Conference, ACM, New York, November.

[69] J.B. Dennis and D.P. Misunas, "A preliminary architecture for a basic data-flow processor," Proc. Second Annual Symp. Computer Architecture, Houston Texas, January 1975, pp. 126- 132.

[70] J.B. Dennis, "Data-Flow Supercomputers," Computer, November, 1980, pp. 48-56.

[71] Guzman, A., "A Parallel Heterarchical Machine for High-Level Language Processing.", in [3], pp. 229-244.

[72] A. Guzman and K.B. Norkin, Final Report of Phase I of the AHR Project, Universidad Nacional Autonoma de Mexico, Technical Report AHR-82-21.

[73] K.E. Batcher, "STARAN Series E," Parallel Processing Symposium, August 1977.

[74] L.C. Higbie, "The OMEN Computers: Associative Array Processors," COMPCON '72, 1972, pp. 287-290.

[75] W.L. Ash and E.H. Sibley, "TRAMP: An Interpretive Associative Processor with Deductive Capabilities," Proc. ACM 23rd National Conference, 1968, pp.143-156.

[76] H.H. Love and D.A. Savitt, "An Iterative-Cell Processor for the ASP Language," in Associative Information Techniques, E.L. Jacks, ed., American Elsevier, New York, 1971, jpp. 147-172.

[77] R.O. Berg et al., "PEPE -- An Overview of Architecture, Operation and Implementation," Proc. National Electronics Conference, IEEE, New York, 1972, pp. 312-317.

[78] J.L. Potter, "The STARAN architecture and its application to image processing and pattern recognition algorithms," National Computer Conference, AFIPS Proceedings, 1978, pp. 1041-1047.